LEARNING PRIMARY MATHEMATICS THROUGH COMPUTER PROGRAMMING

J.B.H. du Boulay

Ph.D. Thesis

University of Edinburgh

1978

ABSTRACT

Many student teachers leave Colleges of Education ill-equipped to teach mathematics in primary school. This study examined the claim that students might learn to understand and enjoy mathematics through learning how to program.

Volunteers with low mathematical qualifications were recruited from a College of Education. They were taught to program in LOGO, an interactive, procedural language containing primitives for drawing, and they undertook programming projects determined by their mathematical difficulties. Three problems were investigated.

(i) The mathematical difficulties of the students.

Observation of students' lessons showed that they did not understand, and had difficulty in teaching, simple arithmetical processes which they knew how to perform by applying rules blindly.

(ii) The difficulties students faced in learning to program.

An analysis of error messages indicated that the students learned fairly easily how to write simple programs to draw pictures in their study of geometry. But certain concepts e.g. variables were difficult for them to learn. Easy access to the computer and concentration on drawing tended to discourage the systematic planning necessary for other classes of problem.

(iii) The effects of the programming on the students' understanding of mathematics.

Case studies of the projects undertaken by the students demonstrated advantages and disadvantages in learning mathematics

through programming. Mathematics became an exploratory activity involving problem-solving and personal discovery. Programming provided concrete illustration of key concepts, e.g. transformations. Students were able to reduce some of their mathematical difficulties and had the satisfaction of success in mathematics. But programming was a complex skill to learn and the mathematics of a problem was sometimes submerged under layers of programming detail, thus presenting the problem at the wrong level of representation. Writing computer programs, which merely embodied rules which the students already knew how to apply, did little to enhance the meaning of those rules.

ACKNOWLEDGEMENTS

I would like to thank the very many people who helped me complete this thesis. Jim Howe, my supervisor, gave me the opportunity to start my research and guided me through it. Many of the ideas in the thesis came from discussions with Tim O'Shea, to whom I am much indebted. Members of the Department of Artificial Intelligence provided a stimulating environment in which to learn how to undertake research. Mark Adler, Yakov Amit, Frances Cassels, Bill Clocksin, Max Clowes, Fran Plane, Mike Sharples, Mike Shneier, Sylvia Weir and Richard Young commented on a draft of this document. Colin McArthur wrote and maintained LOGO, and John Kemplay helped build and maintain equipment. Ailsa Anderson assisted at some LOGO sessions.

Frazer Paxton, Robert Finlayson, Ian Lusk and other staff members of Moray House College of Education allowed me access to their students, lent me equipment, advised me and gave me introductions to Head-teachers. The Head-teachers of many primary schools in the Lothian Region and in Fife gave me permission to observe student teachers while they were on teaching practice. Without the willing cooperation of the anonymous student teachers, there would have been no research. The Social Science Research Council supported me financially.

My greatest debt is to my wife and children who have put up for so long with someone who was 'doing his thesis'.

CONTENTS

TABLE OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 OVERVIEW

A significant proportion of student teachers do not like or understand the mathematics they are to teach in Primary School. They can enter Colleges of Education, gain their teaching Diplomas and return to school to teach mathematics with meagre mathematical qualifications and with little improvement in their mathematical ability and attitude (Rees, 1974; Haylock, 1977; Lumb, 1974). It is unfortunate that such students can be responsible for the foundations of children's mathematical education. Increasing student teachers' understanding of mathematics is a necessary condition of improving the way mathematics is taught in Primary Schools. This thesis has investigated one method by which such students may be helped.

Over the last few years increasing use has been made of computers in the teaching of mathematics. They have been used at all levels from primary school to university, and in a great variety of roles. The computer has been employed to provide interactive drill and practice (Tait et al., 1973), as a management aid for the human teacher (Leeson and Jaworski, 1975), and as a resource for students to use (Dwyer, 1975). It is this last role that is examined here, that is, the computer used as a resource. In particular this thesis has investigated the claim, argued most strongly by Papert (1973), that certain kinds of programming experience increase students' understanding of mathematics and improve their attitude to the subject. He writes:

"Most people emerge from high school without ever having had a

> joyful or personally meaningful mathematical experience. No wonder they hate it and refuse to learn it! We think it is important and easy to remedy this for college students in academic trouble, for future teachers (especially)...."
>
> (p.36, his emphasis)

The objective of this study was to examine how student teachers, with the bare minimum of qualifications, might learn mathematics through their experience of computer programming. This study makes three contributions. Firstly, it identifies a number of difficulties encountered by students when teaching arithmetic. Secondly, it provides a realistic account of the advantages and disadvantages of learning mathematics through programming. Thirdly, it describes a strategy for teaching interactive programming which has been incorporated in a programming primer, and it provides data about the way novices learn to program.

There is no obvious link between learning mathematics and writing computer programs. However, the activity of programming has two important properties. Firstly, programming is directed toward producing explicit descriptions of processes in the given programming language. Secondly, in certain interactive computer systems, the programmer can see some of the processes in action and examine their effects. We will argue here that programming can provide an unique opportunity for the learner to express, specify and investigate a variety of mathematical processes and structures. Feurzeig et al. (1969) give a clear statement of the claims associated with a particular programming language called LOGO. LOGO is a simple, interactive procedural language derived from LISP (McCarthy, 1969). It has primitives for data manipulation and to control a wide variety

of peripheral devices. Briefly the claims of Feurzeig et al. are:-

(i) Programming provides some justification for, and illustration of, formal mathematical rigour.

(ii) Programming enables mathematics to be studied through exploratory activity.

(iii) Programming gives insight into key mathematical concepts.

(iv) Programming provides a context for problem solving and a language with which the student may describe his own problem solving.

We will provide evidence in favour these claims, especially those about the opportunities for mathematical exploration and for the study of key concepts. But we will also show that there are two major drawbacks, apart from cost, to using programming. First, the skill of programming itself needs to be learned and this can prove hard. Second, concentration on the programming aspects of a problem can obscure what is important mathematically.

A number of students were recruited, as volunteers, from a local Scottish College of Education. They were taught mathematics through their experience of computer programming. In order to make the study relevant to their immediate needs, the mathematics work was based in those areas in which they were weak or were having difficulty. Thus in this investigation of the mathematical effects of the students' computer programming experience, three major sub-problems were tackled:-

(i) The students' mathematical difficulties.

(ii) The difficulties arising from teaching the students how to program.

(iii) The effects of their programming experience on their

understanding of mathematics.

(i) Mathematical Difficulties

Various techniques were used to determine the mathematical difficulties of the students. The most important of these was the observation and audio-recording of lessons taught by the students and the subsequent discussions with the individual students about those lessons. These discussions were also audio-recorded. This study was particularly concerned with the way the mathematical difficulties of the students affected their teaching of mathematics. This is why the study relied predominantly on observation and discussion of lessons rather than on psychometric techniques.

The students had a disturbing fear and dislike of mathematics and they lacked competence in even elementary parts of primary school arithmetic. But the students most pressing difficulty was their inability to explain the meaning of mathematical rules which they knew very well how to apply. Observation of lessons showed how this lack of understanding was reflected in poor explanations to the children. This demonstrated that a student's ability to perform a mathematical algorithm was not a reliable guide to her ability to teach the concepts embodied in that algorithm.

A mathematics test was administered. This showed that the performance of these students was comparable with the performance of a broader population of student teachers (Rees, 1974).

(ii) Programming Difficulties

The students learned the programming language LOGO, originally developed for children by Feurzeig et al. (1969). Most found this enjoyable, though some experienced just the same unpleasant feelings

of frustration which they associated with mathematics. The students typed commands to a variety of computer controlled drawing devices. With these devices, the students were able to produce many geometric and other drawings and were able to illustrate a number of mathematical processes, for example, integer operations. They also wrote programs to carry out various kinds of calculation and symbol manipulation, for example, to illustrate the process of division. A complete record of the students' interaction with the computer was kept. The students' discussions with the author were audio-recorded.

Analysis of the students' coding errors showed that they made the same kind of mistakes as children (e.g. those studied by Cannara, 1976). A small number of error messages accounted for the vast majority of all errors generated. Students made many typing mistakes, especially in their use of the 'space' character. They mistook the computational context, e.g. by trying to edit while at command level. Their most persistent mistakes concerned the use of variables, especially the mechanism of argument binding. Although attempts were made to teach top-down problem-solving techniques such as problem decomposition, students often adopted bottom-up techniques when drawing pictures by concentrating on low-level detail rather than on an overall plan. These techniques were inappropriate for other classes of problem, such as those involving symbol manipulation. In general, students had difficulty in writing programs of any complexity. This limited the kind of program which they could be asked to write.

(iii) The Effects of Programming

The students worked through individual mathematics worksheets to

investigate a wide variety of topics. The content of these worksheets was generally determined by the students' mathematical difficulties. Typically a worksheet invited the student to write and debug a program to illustrate a mathematical idea. Occasionally an idea was explored by reference to a hypothetical program which was planned, but not run on the machine. Later worksheets asked students to run and observe the behaviour of primitives provided for them in the domain in question. The students were encouraged to disclose and discuss their mathematical difficulties during the programming sessions. All the work undertaken by the students was documented and the discussions between the students and the author were audio recorded.

Students reacted to the task of learning mathematics through programming in different ways. Programming did provide opportunities for personal mathematical discoveries and excitement, as Papert has argued. But often the programming details were intrusive and deflected the students' attention from mathematical issues. For instance, this happened when the student became more interested in programming problems, which they usually, though not always, found fascinating. It also happened when the mathematical problem was obscured by the levels of programming detail needed to solve it. That is to say, when the mathematics was tackled at the wrong level of representation. For example, if the student was asked to write procedures which drew a picture illustrating a mathematical idea, e.g. a fraction pie-chart, she concentrated on the details of drawing the picture rather than on the underlying mathematical idea which the picture was supposed to illustrate. Ease of access to the computer encouraged the students to adopt trial and error techniques

rather than planning and analysis. This was related to the students' concentration on the picture 'products' of their work and to their failure to investigate the geometric properties of the pictures.

The author was able to minimise the intrusive nature of programming. Special primitives were provided (written in LOGO), which the students used to tackle particular mathematical issues. Students observed the effect of running these primitives and their interaction with each other. This freed the students from the time consuming (and often too difficult) task of constructing such primitives for themselves.

1.2 LAYOUT OF THE THESIS

The next chapter describes the context of the study. It examines previous investigations of the mathematical competence and attitudes of student teachers. These studies have shown that many student teachers perform badly in written mathematics tests, even after taking professional courses in Colleges of Education. Attitude tests have demonstrated that many student teachers dislike mathematics and that professional courses do not always improve these attitudes.

The claims made for the value of learning mathematics through programming are then set out and analysed. Particular attention is paid to claims about LOGO, the programming langugae used in this study. Previous attempts to verify these claims are criticised. Most have failed to substantiate the claims made. In particular, the difficulties of learning programming and of balancing the mathematical and programming content of the work have been given little prominence. A strategy for teaching LOGO is argued and

previous research into novices' difficulties with this language are summarised.

Chapter 3 gives an overview of the experiment. It describes how the three sub-problems, set out above, were investigated. It includes a description of LOGO and outlines the content of the worksheets employed to teach programming and mathematics.

Chapter 4 describes the mathematical difficulties of the students. Using transcripts from lessons and discussions, it shows how they had difficulty in explaining mathematical rules which they knew how to apply. Examples are given of the students' marked lack of mathematical self-confidence. The students' answers to a mathematical test are analysed and compared to Rees' (1974) larger group.

Chapter 5 sets out the experience of the students as they learned to program and compares them to the children studied by Cannara (1976). Their coding errors are tabulated by examining the error messages which their programs generated. An account is given of the planning and debugging strategies adopted by the students. An annotated protocol covering eight sessions exemplifies the problems which students faced in writing complex programs. It shows how a planning strategy derived from constructing programs to draw pictures does not work satisfactorily for a program to manipulate symbols.

Three case studies are used to show how the students learned mathematics through programming. Chapter 6 gives the most detailed study, that of Jane. This shows how she used programming to investigate a number of topics with which she had been having difficulty e.g. angles. Her attitude to these topics improved, but overall she remained pessimistic about her ability to understand

mathematics. Successful and unsuccessful projects are described. Projects were unsuccessful when Jane concentrated on programming rather than on mathematical issues.

The contrasting work of Irene and Mary is described in chapter 7. Irene found programming most unpleasant and gained little benefit from it. Like Jane she did not often see beyond immediate programming concerns to the underlying mathematics. Mary worked with much more self-confidence than either Jane or Irene. She was able to use the programming as a aid to study mathematics. She made a number of personal mathematical discoveries which are described using excerpts from her conversations with the author. These case-studies provide evidence which supports many of the claims made for programming. But they show that this method of learning mathematics has a number of severe disadvantages.

Finally chapter 8 summarises the contribution of the thesis and concludes with some suggestions for further work.

CHAPTER 2

MATHEMATICS AND PROGRAMMING

There is an acute problem in the provision of teachers for Primary Schools who are adequately trained in mathematics. Many student teachers perform badly in mathematics tests and have negative attitudes towards mathematics, even after their professional training. This indicates that reforms are needed in the way student teachers learn mathematics in Colleges of Education.

Papert has claimed that programming in LOGO offers a solution to this kind of problem. This chapter describes the difficulties which teachers have with mathematics and analyses the case for learning mathematics through programming. It also examines the difficulties which students face in learning to program and sets out a strategy for teaching programming.

2.1 MATHEMATICAL DIFFICULTIES

Several studies have demonstrated that student teachers perform badly in mathematics tests. Their poor results extend even to elementary parts of arithmetic. Attitude tests have shown that student teachers often dislike the subject, and some continue to dislike it after completing their professional training.

Performance in Mathematics Tests

Rees (1974) tested 108 student teachers at the end of their second year of study. These students had completed a mathematics course and had taken a mathematics exam within the College of Education. Rees remarks that they "could therefore be said to be at the peak of their performance." She found, however, that these

students did not score highly. Their mean score was worse than that of a sample of school pupils who had taken G.C.E. 'O' level mathematics but better than a sample who had taken the C.S.E. examination (Certificate of Secondary Education, of lower standard than G.C.E.). Rees identified a "common core" of items from her test which were answered incorrectly by more than half those tested. These included a number of elementary fraction and decimal manipulations. Further details of Rees' test are given in chapter 4.

In another study, Haylock (1977) tested 182 first-year College of Education students using the Bristol Achievement Test. These students had already taken an introductory mathematics course and had gained some experience in schools. Haylock found that "one quarter of the students scored low enough to be bettered by more than 10 per cent of 12 year olds." He noted particular weakness in the sections on reasoning and also on number and arithmetic processes. This latter supports Rees' findings.

A similar study was conducted by Lumb (1974). He tested 296 students and also found weakness in number and in arithmetic processes. Lumb used questions taken from the textbooks of the School Mathematics Project. His questions included a number of items related to 'modern' mathematics. He found that these questions were answered very poorly. He notes that "the depths of ignorance of mathematical facts and basic computational skills revealed in the initial test were absolutely staggering." Two interesting facts emerge from this study. First, over half of Lumb's students (55%) had passed G.C.E. 'O' level Mathematics. Secondly, the students were retested at the end of their mathematics course in the College but still showed evidence of mathematical weakness. There was some

improvement in test scores but still many students made computational errors. For instance, 58% still could not put five fractions in order of size and 1% still thought that '7 - 2 = 2 - 7'. He also noted that a number of students confused multiplication and division when dealing with fractions.

Despite the fact that these three studies used different tests, they all suggest that many students are weak in arithmetic. (Similar weaknesses have been found in the arithmetic ability of Elementary Teachers in the U.S.A, see for example Eisenberg, 1976.) All these studies tested the students after they had taken some part of their College mathematics course. Haylock suggests that the minimum entry qualification for Colleges of Education should be G.C.E. 'O' level mathematics. While this may help a little, Lumb's findings suggest that this is not necessarily a guarantee of mathematical competence. Unfortunately Lumb did not break down his test results into those with, and those without 'O' level mathematics. He merely indicated the proportion of those tested who had 'O' level. From the figures we can infer that some of those who had passed 'O' level mathematics had difficulty with elementary arithmetic processes. Recent concern over the mathematical competence of student teachers has led to a number of calls to introduce a minimum entry qualification (see the Joint Mathematical Council of the United Kingdom, 1977). Matthews and Bajpai (1977) doubt whether this will be of much benefit and fear that introducing this extra hurdle for aspiring teachers may actually increase their dislike of the subject. These results demonstrate that not only do student teachers do badly in mathematics tests but that professional courses in Colleges of Education do not always improve matters very much. Raising the entry qualification will not

entirely solve the problem.

Attitudes to Mathematics

There is evidence that many student teachers enter Colleges of Education with negative attitudes to mathematics. Some Colleges are able to improve these attitudes, others are less successful. Rees (1974) in her study of student teachers asked them to underline the word which best described their feelings about mathematics, from the choices: "I like/dislike/tolerate mathematics". Out of the 107 asked, 44 underlined 'like', 41 underlined 'tolerate' and 22 underlined 'dislike'. As one might expect, the scores in the mathematics test correlated with this result so that the scores were:

'likers' > 'tolerators' > 'dislikers'.

Anecdotal evidence of students' attitudes is offered by Kerslake (1974). She stresses the need for Colleges of Education to break down the 'antipathy' of students to mathematics which she has encountered. Ray (1975) has shown that this can be done. He carried out a factor analysis on a questionnaire given early in the first year and then again in the second year to students at a College of Education. The questionnaire sought to assess the attitude to mathematics and the attitude to the teaching of mathematics among the students. He found that 'general' attitude to mathematics did improve but that primary students were the least favourably disposed to the subject. Disappointingly he found that only students specialising in mathematics improved their attitude to teaching mathematics (which he distinguished from their attitude to the subject itself) and to the newer methods.

A similar investigation of changes of attitude among student

teachers was conducted by Lumb and Child (1976). They also found improvements in attitude, but in their case the largest improvement was among primary students. They administered two tests to measure students' attitudes to mathematics and to the teaching of mathematics. One test was a semantic differential and the other was a questionnaire similar to that used by Ray. These two tests were administered to 287 students on entry to a College of Education and again after the students had completed two mathematics courses, about two years later. All students spent their first year in a common course. Then they took a course appropriate to the kind of school they wished to teach in: primary, middle or secondary. The students' responses to the tests were grouped by sex and by type of school preference. No significant differences were found between students on entry to the College. At the end of the courses, those students who had opted to teach in primary school had much more positive attitudes relative to their initial attitudes than any other group. The inference is that students whose attitudes became more positive took better courses. But as Lumb and Child admit, the cause for the difference in attitude was not established.

Colleges of Education are aware of the difficulties which students have with mathematics. The Mathematics Section of the Association of Teachers in Colleges and Departments of Education (1973) note gloomily:

> "There is strong feeling that the young teachers we are sending out are unable to meet the challenge of the new approaches and are, moreover, less able to cope with the traditional, more formal methods. We admit some truth in the charge, yet the problems faced are immense. Of students admitted to the

Colleges in 1968, 27.5 per cent of the men and 44.6 per cent of the women lacked even the elementary qualification of 'O' -level mathematics. But no statistics can show the sheer lack of mathematical understanding or the incidence of dislike and even fear." (p.123)

These studies demonstrate the seriousness of the problem of student teachers and their mathematics. They also indicate that any proposed solution is likely to face severe difficulties.

2.2 THE VALUE OF PROGRAMMING

This section sets out the case for teaching mathematics through programming. Papert has been a persuasive proponent of the value of LOGO programming as a means of exploring mathematics. This study has used many of his ideas and was much influenced by his claims. He argues that programming can improve students' attitudes to mathematics by giving them the enjoyable experience of doing mathematics, which may well be unfamiliar to them.

The specific claims made for the value of LOGO programming have been set out clearly by Feurzeig, Papert et al. (1969). They make four kinds of claim:-

(i) Programming provides some justification for, and illustration of, formal mathematical rigour.

(ii) Programming enables mathematics to be studied through exploratory activity.

(iii) Programming gives insight into key mathematical concepts.

(iv) Programming provides a context for problem solving and a language with which the student can describe his own problem solving.

The first three claims have been made for languages other than LOGO (see for example School Mathematics Project, 1974). There is evidence that programming in BASIC can improve numerical ability and attitudes to mathematics, especially amongst the less able (Bjork, 1975). The fourth claim about providing a descriptive language for problem solving is particularly associated with Papert (and with LOGO). This claim has received the most attention in the LOGO literature.

LOGO has a number of features which make it a congenial language to use in the study of mathematics. It is interactive which means that a student gets immediate information about his program. It is procedural. This enables programming (and mathematical) problems to be decomposed into sub-problems, where each sub-problem is solved by a separate sub-procedure. By defining and naming procedures, the user is able to extend the language. This feature distinguishes it from BASIC. Most implementations of LOGO have primitives which drive drawing devices. The drawings produced by programs act as 'traces' of those programs, as motivation for the student and enable visual, geometric topics to be explored. The study of the properties of these drawings is known as "Turtle Geometry" (Papert, 1972). A more detailed description of LOGO is given in the section 3.3 of chapter 3.

(i) Formal Mathematical Rigour

Feurzeig and Papert argue that many people find the rigour demanded in mathematical expression either constricting, unintelligible or perverse. It is not seen as as a means of reducing ambiguity either for communication between people or for one person

communicating with himself. In contrast, communication between a person and a computer is dependent on the person framing unambiguous instructions for the computer to execute. Most people, new to programming, accept that computers have to be addressed using formal language, and will come to see the value of such formality. The advantage of LOGO programming is that even poorly formed expressions, or an ill-thought out sequence of instructions will usually produce some observable effect. Either the computer will be unable to interpret the expression, or the sequence of actions it carries out will not be as expected. The advantage of symbols having unambiguous meanings is that the user can plan precisely what he wants the computer to do and can interpret the causes of the actions it has carried out. The value of this rigour is that it forces the user to produce an explicit description of the process to be carried out. Working with paper and pencil, in mathematics, the user may commit all kinds of errors which may not become apparent to him. This is because he must both specify the instructions and execute them himself.

Some support for this claim is provided by Howe and O'Shea (1976). They are investigating the effects of LOGO programming on the school mathematics of Scottish primary school children (aged about 12 years old). They describe how some of the children exhibit marked gains in self-confidence in their school mathematics and become more 'mathematically argumentative'. This implies a search for rigour by the children.

Naturally the formal properties of the programming language have to be learnt, as they have to be in mathematics, and this may involve the user in considerable effort. It could also be argued that

programming with its need for rigour and precision of expression might inhibit intuitive and abbreviated forms of mathematical thinking (as described by, for example, Kruteskii, 1976). This is partially answered by Feurzeig and Papert who argue that programming experience provides an excellent example of the distinction between "the global planning of an attack on a problem and the formal detail of an elaborated solution." The former may be only a partially specified sequence of major actions to be carried out, while the latter is a complete working program. The first analysis of the problem may well be intuitive, producing an initial plan of action, which is then gradually refined using more formal methods into the precise program.

(ii) Mathematics as Exploratory Activity

Feurzeig and Papert explain how peripheral devices controlled via a suitable language can function as a mathematical laboratory in which mathematical exploration is possible. In this way mathematics can be made an enjoyable activity. An interesting set of such mathematical laboratories is described by Dwyer (1975), based on programming in an extended version of Basic. He has implemented a Computer Lab., a Dynamics Lab., a Logical Design Lab., a Synthesis Lab. and a Modelling/Simulation Lab. These enable a diverse set of mathematical applications to be explored e.g. algorithms, time dependent processes and simulations.

These claims for 'applied' mathematics must be seen as partly a reaction against the different tradition in school mathematics in the U.S.A. which has tended to concentrate rather more on 'pure' mathematics. Contemporary British primary school mathematics books

abound in descriptions of activities for children (see for example Association of Teachers of Mathematics, 1969). Indeed the use of such exploratory activity might be said to be the dominant theme underlying British primary mathematics education. Recent American modern mathematics projects have been dominated by their universities in contrast to the high teacher participation in British modern mathematics projects (Griffiths and Howson, 1974). This has led to criticism that such courses were too abstract and formal to be taught successfully at school level (see e.g. Minsky, 1970). Work with the computer is seen as a method of counteracting this criticism. The student can set up a mathematical process and may be able to observe it running. The computer system can be designed to embody a particular mathematical system, e.g. a geometry, which the student can explore (Papert, 1972). This principle of embodying a mathematical system in a piece of apparatus is not confined to computers. Cuisenaire rods and Dienes multibase arithmetic blocks are two systems for exploring aspects of arithmetic and numeration systems. Both Papert and Dienes argue that the idea of a theorem arises naturally out of the activities of children using their two mathematical embodiments. Papert describes the "Total Turtle Trip Theorem" from his "Turtle Geometry" (see section 3.3 of chapter 3). He argues that this is a theorem children can both discover and make good use of to solve drawing problems. Dienes (1973) shows how children may progress from games played with his Logic Blocks to the proof of theorems about the isometries of the equilateral triangle.

The particular value of programming is that both objects and operations can be modelled. The distinction between a state and a transformation can be made explicit. Hand cranked calculators can

model the operations of basic arithmetic and illustrate the relations between those operations. Rotations of the machine's crank show how subtraction is the inverse of addition and that multiplication is repeated addition. But the machine lacks the versatility of a computer, which can be programmed to illustrate these and many other mathematical operations and relations between operations (Williams (1971) reviews the use of apparatus in primary school mathematics).

A second benefit of basing mathematics work on programming is that certain important mathematical skills can be practised. These include 'problem-posing', 'problem-solving' and 'generalising'. This latter is made possible because the student himself can extend the solution of a particular programming problem to the more general case. Thus, for example, the underlying similarity of LOGO procedures for drawing an equilateral triangle, a square and a regular hexagon can be exploited to produce a general procedure which will draw any regular polygon. By reference to the sequence of events which led from writing the individual polygon procedures to the general polygon procedure, the nature of the activity of generalisation can be illustrated. Milner (1973) taught children about 'variables' through their experience of programming. He describes the opportunities which this work gave the children for peer-teaching, problem-solving, indulgence in mathematical curiosity and for generalisation of a solution of a particular problem.

The School Mathematics Project (1974) suggests that, by writing programs, students can represent processes which would be insoluble (for them) by any other means. They give examples of "a car braking, a child collecting gift cards, an epidemic spreading or the simple processes leading to Pascal's triangle". Related claims are made for

the MATLAB system used by engineering students who would otherwise find the mathematical computations needed in their course either too tedious or impossible (Hampton, 1976). In this case the student runs pre-written programs, which free him to explore a mathematical application, by undertaking the mundane calculations. The student is not expected to write programs himself.

(iii) Key Mathematical Topics

Depending on the language and the particular peripheral devices in the system, a number of key mathematical concepts can be illustrated. Two concepts, mentioned by Feurzeig and Papert in connection with LOGO, are those of 'function' and 'variable'. They argue that in programming a number of issues arise concretely including:

> "the many roles of 'X' in algebra: sometimes it appears to be a number, sometimes a subtley different kind of object called a variable, and other occasions it is to be treated as a function." (p.7, their emphasis)

Three studies have attempted to evaluate this claim. Feurzeig and Papert (1969) taught LOGO programming to twelve mathematically average children aged between seven and nine years old. The children learned to write and debug simple procedures. The researchers were mainly concerned to assess the difficulties of teaching such young children how to program. They concluded that they had been successful. They also claimed that "children of this age do acquire a meaningful undertanding of concepts like variable, function and formal procedure (though not in those words) through their experience with programming."

Another study was conducted by the same researchers using slightly older children from a Junior High School. There were twelve seventh-grade children (13 years old) who were in the median range of mathematical performance. This study was much more closely concerned with teaching children mathematics through programming. The children were taught arithmetic and algebra by the researchers instead of being taught by their school teachers. The study lasted about a year and the mathematics was introduced and explained through programming concepts. The report gives details of the work on 'functions' and 'variables' undertaken by the children. Results of the evaluation were inconclusive. The control group gained slightly more than the experimental group in that part of a post-test concerned with arithmetic problems (the Iowa Test of Basic Skills). Doubt was expressed that the test was measuring the skills and concepts transmitted by the programming. Invited observers and the children's teachers were generally favourable about the effects of the course.

Milner (1973) taught children programming and investigated the effects of this experience on their understanding of 'variables'. He taught 18 fifth-grade students who were selected at random. The students were taught how to program, including the construction of recursive procedures. These procedures were used to generate and print number series. A control group was taught nothing about variables. The pre and post-test items asked the children to evaluate expressions containing variables whose values were given, or to find the value of variables satisfying given constraints. The experimental group made significant gains in test scores. The control made no gains and this was taken as a measure of reliability of the test. Milner argues that the procedures written by the

children and their explanations of the actions of the procedures provide convincing evidence of the childrens knowledge of variables. He cites the case that some children used numerals as variable names and were apparently able to distinguish between the name and the value. The problem of how the programming work with variables was, or should be, integrated into the broader school mathematics curriculum was not addressed at any length.

In Papert's "Turtle Geometry" a number of other key concepts are also given vivid illustration. In particular translation, rotation, state, state change operator, angle as rotation are all used by the students in the course of constructing a variety of pleasing plane figures.

(iv) A Context and Language for Problem Solving

The above claims state, in essence, that it is possible to exploit the activity of programming in a mathematically interesting way, and that it is possible to implement explorable mathematical systems as computer programs. But Feurzeig and Papert go further. They argue that programming gives the pupil many opportunities to solve problems, often of his own devising. The pupil can be given insight into his own problem solving processes by using the record of his dialogue with the machine as an indication of his own thinking.

The underlying assumption here is that there are useful general problem-solving methods to be learned from programming which can be applied in other domains such as mathematics. They argue that problem-solving methods emerge much more easily out of programming activity than out of mathematics. This is because a plan for a program can and usually is written down, and the changes which take

place in this plan can be monitored as it turns into a working program. By contrast mathematical problem-solving often yields only sparse documentary evidence of the process of solution unless special steps are taken. In programming the idea of 'sub-problem' and 'sub-goal' can be mirrored in identifiable programming constructs e.g. sub-procedures. The idea of 'debugging' and the value of 'mistakes' comes out clearly. In this way programming can provide both a context in which a variety of problems may be posed and solved as well as a language for describing the problem-solving process.

Statz (1973) tested specific hypotheses related to the effects of programming on problem-solving. She taught programming and problem-solving to an experimental group of children (aged from 9 to 11 years) for a year. Sixteen children were taught in their school, in school time. Statz developed a model of problem-solving, influenced by Polya (1957), and related features of this model to specific programming activities.

Statz's first hypothesis was that the children who had learned to program would perform better on a battery of four problem-solving tasks than a control group, who were not taught programming. Significant gains of score were found on only two of the four tasks. One task consisted of a series of word puzzles involving anagrams and word classification. The other task asked the children to find permutations of three or four digits. Statz argues that the classification skills needed in the first task and the ability to isolate and control variables needed in the permutation task were both developed by the programming experience. The two tasks for which no significant gains in test score were observed both involved the use of "strategy" which would not have developed as a result of

programming experience. One of these tasks was the Tower of Hanoi puzzle with five rings. Each child was assigned a score which was derived from the ratio of the number of moves he took compared to the theoretical minimum number of moves to transfer five rings. Thus a child who did not perceive the structure of the problem correctly at his first and only attempt would possibly score badly. Anzai (1977) has shown that adults can take several attempts before they perceive the structure of this problem clearly enough to produce the minimum solution confidently. Statz might have observed differences in the way the experimental group improved their solution, over a number of trials, compared to the control group. This would fit better with those claims for programming which say that it helps one to plan and gradually reformulate plans. A number of other criticisms have also been made by Weyer and Cannara (1975) about the assumptions underlying Statz's statistical analysis. They question the validity of assigning scores to the kind of task set by Statz.

Statz's result does not entirely rule out the criticism that in learning programming a student learns problem-solving skills relevant only to programming which cannot be transferred to other domains such as mathematics. For example, Polya (1957) gives excellent advice on how to solve a variety of mathematical problems. But without specific mathematical knowledge one cannot make use of even his first heuristic which is 'understand the problem'. The student might well learn that problem decomposition is a good heuristic but still not know how to decompose a given mathematics problem. Feurzeig and Papert partially answer this criticism as follows. They argue that programming provides a 'natural context' to 'concretize' such advice and that:

> "It is at least highly plausible that pupils who have acquired very early the habit of organising their approach to a mathematical problem (through programming) will be better able to develop systematic habits of thought in the more murky areas of problem-solving they will have to meet later, in school and elsewhere."(p.8)

Papert (1971) develops this point more forcefully. He claims that children (and presumably adults too) often have counter-productive theories about cognitive functions such as problem-solving, which he has named the "Pop-ed culture". He gives the example of children who believe that the best way to memorise something is to make one's mind 'blank'. His argument is that any more principled advice about memorisation or problem-solving is bound to be beneficial because it is almost certain to replace the child's existing and incorrect theories about how he thinks or how he ought to solve problems. Not only does Papert (1972) wish to give children insight into their own thinking, he also wishes to change the emphasis in mathematics education away from teaching particular pieces of mathematics towards teaching the activity of doing mathematics.

As Papert argues, problem-solving is an important part of 'doing mathematics'. In a paper with Goldstein (1976), he explains how awareness of problem-solving concepts and use of the appropriate descriptive language could help a teacher deal with a child's arithmetic difficulty. Their argument over-emphasises the role which a language for problem-solving might play at the expense of knowledge of arithmetic. They imply that knowledge of heuristics can replace knowledge of arithmetic. They present the example of a child who had

written the following sum:-

$$\begin{array}{r} 35 \\ +35 \\ \hline 610 \end{array}$$

They comment that "the teacher involved had little to say besides observing that the answer was wrong"(p.67). They hypothesise that the child had followed:

> "the reasonable linear plan of assuming that the sum of multi-digit numbers could be achieved by adding the columns independently"

and that in "debugging" this misconception he must take account of the

> "interaction through ordering the columnar additions right to left and utilizing a "carry" data structure"(p.68, their emphasis)

This theoretical analysis uses a descriptive language derived from programming. Davis (1977) has conducted similar analyses of children's mathematics to good effect. He showed how a child had a set of consistent though incorrect sub-procedures for carrying out fraction to decimal manipulations. But Goldstein and Papert go further. They wish to teach this vocabulary to the children through their programming experience:-

> "...given access to a vocabulary for programs, plans and bugs, we believe that student and teacher could be articulate about describing the particular algorithm used by the student in reaching the '610' answer, in identifying the bugs, and in debugging the addition program (in the student's head) to yield the correct result."

Goldstein and Papert should have also stated that the

interaction of column addition is a piece of arithmetic knowledge, which the teacher should have been able to deal with. Such an analysis of the student's "bug" will only make sense if the student understands the arithmetical nature of his error. That is to say there are two levels of representation of the student's difficulty: one about arithmetic and the other about heuristics. Without the arithmetic, the heuristics cannot be applied. It is necessary that the bug be presented to the student at the appropriate level of representation. We would argue that "access to a vocabulary for program plans and bugs" can enrich but not replace arithmetic knowledge.

Most of the LOGO programming studies with teachers (and undergraduates) have concerned themselves with problem-solving. A number of groups of teachers have been taught programming as an aid to learning mathematics. The largest number has been in Quebec, where nearly five hundred students, destined to become mathematics teachers have been taught programming as part of their course. Arcouet (1976) estimates that about one third of the secondary school mathematics teachers in Quebec will have learned to program in LOGO.

The evaluators of the effects of this widespread introduction of programming have largely restricted themselves to an evaluation of the way the course was received by the students (Daniel and Villardier, 1976). Students learned to program at remote terminals. Many of the questions in the evaluative questionnaire concerned the students' difficulties in learning to program and in communicating with the main centre. As well as learning the elements of LOGO, the students also had to work on a personal project which was to give them a clearer insight into debugging, problem formulation and

solution. Few of the questions in the questionnaire, sent out to one hundred and fifty students, concerned the students' problem-solving. There was one question which asked the students to judge how far the course had fulfilled one of its objectives which was "Ameliorer ses methodes de resolution d'un probleme?". ("To improve your problem-solving methods.") In general students seemed to believe that the course had done some good in helping their problem-solving, though responses were varied.

Statz (1973) taught programming to a group of thirty-six undergraduates, recruited from a mathematics education course. They were to help to teach children in Statz's other study, mentioned earlier. Of these students, some were "frightened away by initial problems" and others dealt with programming in a "pedantic fashion", following examples from the manual and the class but without much personal involvement or exploration. A number did become enthusiastic.

As a result of this experiment a thirty-hour, in-service summer workshop was organised for sixteen teachers from schools in and around Syracuse. Only two of the participants had any previous programming experience. The teachers wrote a number of programs and took part in discussions concerned with the links between mathematics and programming. At the start of the workshop only three were considered to have "some acquaintance with a model of problem solving". At the end about 60% of the participants were able to outline a model for problem-solving derived from their programming experience.

Austin (1976) has run a number of courses for teachers. In all about thirty volunteers from a teacher's college were taught to

program and undertook a number of individual programming projects. They worked in a very well equipped programming classroom complete with a wide variety of computer driven peripherals including a music box, phoneme generator and drawing devices. One objective of the course was to acquaint the students with a descriptive language for problem solving. They were shown the use of such terms as "debugging" and "sub-procedurisation" both through programming and also by being taught to juggle. This latter skill was chosen to show the power of such terms to describe a wide variety of processes. Despite their initial scepticism, the students were taught to juggle easily by helping them to break down the complex juggling movements into separate learnable sub-movements. The second objective of the course was to produce an experimental group who could be studied in their turn teaching children the same skills. This second phase has not yet been reported.

Feurzeig and Lukas (1971) taught LOGO to a group of professional teachers, professors of education and staff of Bolt Beranek and Newman. Seven of the nine participants had no previous experience of programming. Two elementary teachers in the group were judged to have limited mathematical backgrounds. The objective of the course was to give the participants the opportunity to formulate a theory of problem-solving derived from the programming. The researchers describe how particular heursitics, such as 'find the easy cases of the problem and then reduce the hard cases to the easy cases', could be exemplified by writing computer programs. No formal evaluation of the work of the participants was undertaken.

A course was also run, by the same researchers, for nine randomly chosen undergraduates. These students had scored badly in

their College Entrance examination and would need extra help to pass a mathematics course requirement (Feurzeig and Lukas, 1971). The report gives only sparse details of the mathematical difficulties of these students and did not evaluate how far the programming had helped them.

None of the above studies has related the programming work of the participants to their particular mathematical needs as teachers. Elliott (1976) reviews the claims for teaching programming to teachers and to student teachers (and she includes versions of claims (i) to (iv) above). Elliott describes how the computer can act as a surrogate pupil to whom the teacher can 'explain' a piece of mathematics in a computer program. She argues that there are similarities between programming and teaching, especially when the latter is viewed as problem-solving. Some of the problem-solving heuristics from programming might be applicable in the classroom e.g. 'teach a complex task by breaking it down into simpler sub-tasks'. It is also useful for the teacher to reflect on the experience of being a <u>learner</u>. Awareness of her experience of learning programming should make the teacher more sensitive to the difficulties of her pupils learning mathematics.

There is an elaboration of this argument, similar to Papert's objective of teaching children to <u>be</u> mathematicians rather than teaching them about mathematics. In programming there is special emphasis on individual mathematical explorations by students. This implies a <u>consultative</u> rather than a <u>managerial</u> role for the student's tutor. His job is more to help the student in her exploration of mathematics than to lecture pieces of mathematics to her. (Caricatures of these two opposing styles of mathematics

teaching are given in Griffiths and Howson, 1974). In this way the cogency of the subject resides not in the authority of the tutor's position but in the subject itself. (For a discussion of this point see Skemp, 1975). Clearly there must still be explanations and suggestions for work from the tutor, but much of the responsibility for what happens in a session is now the student's.

Disadvantages of Programming

Proponents of the value of programming have tended to minimise (or ignore) the disadvantages of this method of learning mathematics. There are two main disadvantages.

(i) Learning Programming

Programming is a complex skill to learn and demands an appreciable effort on the part of the learner. In this way the computer compares badly with other mathematical apparatus which is generally very easy to manipulate. The difficulties of learning programming are described in section 2.3 of this chapter.

(ii) Wrong Level of Representation

Programming may lead the student to think about the problem in hand at the wrong level of representation. Clearly it is insufficient just to teach the student programming and hope that, somehow, something of mathematical value will arise spontaneously. If we want students to progress mathematically as a result of their programming work, they must be helped by a suitable curriculum and with appropriate teaching. Otherwise a valuable mathematical insight can easily be hidden by obtrusive programming detail. Unless this bridge-building help is provided it is very hard to see how students

will learn to ask mathematically interesting questions about what they are doing. The progress from programming to mathematics may be quite complex and require different kinds of help at different stages. This is a similar problem to that faced by others who teach mathematics by getting students to manipulate structured apparatus. Dienes (1973), for example, postulates six stages of abstraction in the process of understanding a mathematical system. The students may start by playing with some piece of structured apparatus. Then via a succession of games they come to understand the structure and constraints built into the apparatus. By representing this structure symbolically the students may then examine it. Later they will be able to examine the properties of the symbolic system itself. Finally the students will be able to formulate theorems in the system under investigation. Dienes suggests how the teacher can help the students at each stage to progress to the next, starting from 'idle' play with the structured apparatus he has designed.

Some claim could be made that the explicit language of programming could provide a suitable ready-made symbolic representation for mathematical processes and structures which would decrease the number of stages through which the student must pass. Once the student had learned this (programming) language he would be able to use it to describe whatever new piece of mathematical structure was under investigation. This, however, suggests a high degree of familiarity with a programming language. Even if we concede that such familiarity is in principle possible then the question arises as to whether LOGO is a suitable language for such representations. Schmitt (1975) has pointed out a number of deficiencies in LOGO as a vehicle for mathematical expression and

proposed an alternative command language called MATHCALC.

A related disadvantage of learning mathematics through programming is that the ease with which algorithms can be defined may distort the teaching syllabus. It is not clear that student teachers need to practice the specification of algorithms (see e.g. Elliott, 1978) since they often know them but do not understand them. It is hard to see how the meaning of an algorithm can emerge merely through formalising it in a programming language.

2.3 LEARNING TO PROGRAM

One objection to teaching student teachers how to program, as part of their mathematics education, is that programming itself may be more difficult to learn and more time consuming than the mathematics it is supposed to reveal. There is at present insufficient evidence to refute this argument in the case of LOGO. The LOGO studies generally give few details of the difficulties encountered by their subjects while learning to program. It is usually not made clear whether students found programming easy or whether their difficulties have not been reported. Notable exceptions are the studies of Cannara (1976) and Weyer and Cannara (1975). This section argues for a particular style of teaching programming and assesses, as far as is possible, the difficulties likely to be faced by the student teachers when they learn to program.

Programming may be divided into four kinds of activity: planning, coding, debugging and reading. In an interactive system such as LOGO, planning, coding and debugging activities may not follow each other in a straightforward linear sequence, e.g.

plan->code->debug. In the course of a single session, the programmer may undertake all these activities as he attends to different parts of his program which may be in varying stages of completion. (Miller and Goldstein (1976) are developing a grammar for the automatic analysis of LOGO protocols in which planning and debugging interact.)

(i) Planning: The programmer makes broad decisions about how the program which is to be written will solve the problem in hand. Such questions as what language to use (if appropriate), what program organisation to adopt, what data representation to construct, will be answered at least partially. This part of the programming process has been studied by Hoc (1977), see (ii) below.

(ii) Coding: The programmer translates his plan into commands in the chosen language. This may be a multi-stage process in which, for instance, gradually more refined statements of the solution are made perhaps using names to stand for as yet undefined sequences of primitive commands. Hoc (1977) has compared programmers of varying levels of experience in their ability to plan and code an algorithm (in COBOL) to control a change-giving ticket machine. He found that beginners tended to formulate solutions for specific instances of the problem rather than a general algorithm. This he attributed to the beginners inexperience of machines which could control their own actions. He also found that beginners translated their plans, expressed in terms of travellers, ticket-machines and coins, inefficiently into COBOL code because they had insufficient understanding of the COBOL virtual machine though they knew the basic terms and syntax of the language. Similar difficulties in understanding general algorithms are reported by Weyer and Cannara

(1975). They analysed the way children filled in a partially specified flow-diagram for a chocolate machine, similar to Hoc's ticket machine. Some of the children misconstrued the flowchart. They saw it as a representation of the actions the machine might take in response to a single specific set of input coins rather than as an algorithm to deal with all possible sets of coins. Adult LOGO beginners, taught by Austin (1976), also had difficulty translating plans into sound code, though they knew the basic terms and syntax of the language. Shneiderman (1977) explicitly formulates a theory of programming comprehension based on such a semantic/syntactic distinction.

(iii) Debugging: Partial or complete programs are tested. This will usually lead to changes being made to the code and sometimes to the general organisation of the program or even to the whole plan of attack on the problem.

Young (1974) compared novices with experienced programmers debugging in a variety of languages. He found that both groups had about the same number of errors in the initial versions of their programs but that experienced programmers could eliminate errors more quickly. He argues that the user should be given as much useful information about his errors as possible.

(iv) Reading: The three activities described above are mainly concerned with the production of programs ab initio. This last activity, 'reading', is of more concern for those who have to read, use or amend programs written by other people or by themselves at some earlier time. This activity may be of interest where a student teacher has to read and comprehend a program provided for her which embodies a mathematical concept.

Shneiderman (1977) argues that comprehension of the function of a program is effectively measured by tasks involving recall of the program. He reported an experiment which showed that experienced programmers were much better than novices in recalling short FORTRAN programs they had studied. But when the program statements were scrambled the experienced programmers fared as badly as the novices. (Shneiderman developed this from a method employed by Chase and Simon (1973) to distinguish novices' perception of chess boards from that of experts.) A different method of assessing comprehension was employed by Green (1977). He measured the time it took for programmers to either trace through a program given the starting conditions, or to deduce which starting conditions would have produced a given outcome.

Novice Programmers

A novice programmer not only has to learn how to carry out the activities mentioned above but he must also learn that these are the activities associated with programming. He will also have to learn what class of problem is appropriately solved by a computer program and how to interpret a computer program or its execution as a solution to a problem.

A programming language gives the novice access to a novel kind of machine, a 'virtual' machine. He must learn how this machine works, how to control it and what kind of task he can make it carry out. Learning a first programming language is thus completely different from learning a further programming language. In the latter case the programmer will already understand what programming is about and can concentrate on the particularities of the new

language. The novice will both be learning a new language and be learning what this language may be used for. An appropriate language for a learner may be uncongenial for an experienced programmer. It may lack constructs or give wordy error messages. Clearly the argument depends, to some extent, on why the novice is learning programming. Charmonman and Ralston (1975) argue that, in the case of computer science undergraduates, choice of first language is not critical because they will have to learn so many languages during their course. Even if such an argument can be sustained for computer scientists, it does not apply to the student teachers of the present study. They learnt a single language as a means rather than as an end in itself. Charmonman and Ralston make the less contentious point that the content of the programming course is of paramount importance.

The introduction to programming may be complicated if more than one formal language is involved. To debug, edit and store a program, the novice will often have to formulate commands in a language different from that in which the program itself is written. Further languages may be needed to assemble or compile his program. This means that the novice is being introduced to a variety of virtual machines with different properties, but all typically accessed through the same terminal. The novice has to decide which of these machines he is communicating with at any particular moment. Further complications arise if the program, written by the user, itself defines a further virtual machine, a parser for example. An interesting example of precisely this difficulty is given in Kahn (1975). The child, whom he was teaching, was unsure whether he was communicating with a parser or with the underlying LOGO in which it

was implemented. In LOGO the distinction between the language for the programs themselves and for editing, debugging and filing is not great.

Another difficulty of learning a first programming language results because most of the actions of the virtual machine will be hidden from the programmer. The eventual, perceivable effects may be related to the original commands by a long chain of cause and effect which the programmer may be unable to reconstruct because of his inexperience.

Three inter-related strategies can be employed to alleviate these difficulties. Firstly, the internal, hidden actions of the computer can be made manifest. Secondly, the novice can be provided with a description, or 'story', about the virtual machine at a level of detail appropriate to the computational events for which he needs explanations. Thirdly, the virtual machine can be made simpler by making the language simple and self-consistent, by reducing the number of languages which have to be employed (e.g. for filing or for editing) and by carefully matching the 'story' to the programming language and to the computational events encountered.

Making Hidden Actions Visible

Papert's 'turtle' is an excellent example of a device for explicating hidden 'flow of control'. This has been described in numerous papers, see for example Papert (1972). The turtle is a small computer-controlled motorised cart with an attached pen. Many LOGO implementations include such a device. It is controlled via commands such as 'FORWARD 100' or 'LEFT 45'. The first command causes the turtle to move forward 100 units in whatever direction it

is currently facing. The command 'LEFT 45' causes the turtle to rotate 45 degrees on the spot anti-clockwise. Appropriate sequences of such commands will cause the turtle to trace out patterns which can be exploited for a variety of mathematical purposes (Papert, 1972). If the turtle's pen is 'down', the turtle will leave a line behind it showing its path like a snail. Because the device gives a visible and permanent record of the commands issued, it serves to explicate the idea of flow of control through the sequence of commands.

It has been shown that simple programming can be learnt by even very young children (6 years old) when the turtle is employed and is controlled via a 'button box' input device (Perlman, 1974). The button box has buttons labelled with the names of primitive commands both to move the turtle and to store sequences of such commands for later execution. A second device, also constructed at Massachusetts Institute of Technology, provides an even more concrete illustration of the definition and execution of sequences of commands. This device, a 'slot-machine' (Hillis, 1975), has an array of slots into which cards bearing commands can be pressed. The procedure defined by such a sequence of cards can be executed. The procedure can be edited simply be replacing one card for another.

While such devices can be used to illustrate many important computational concepts, certain of the computer's actions are still difficult to illustrate. For example, the turtle cannot easily be used to illustrate command parsing, assignment and filing. It has been suggested that a visual display driven by the LOGO interpreter could "graphically simulate some of the computer's own internal workings" (Weyer and Cannara, 1975).

Impressive progress towards this goal has been reported in which the action of the language BASIC, rather than LOGO, is illustrated (Barr et al., 1976). The authors have written an 'intelligent' tutorial program which teaches programming in BASIC and which gives the novice-programmer methods of 'seeing the workings' of the BASIC virtual machine which he is learning to control. One facility is a dynamic representation on a visual display of his program's action based on computer generated flowcharts. Special tracing facilities are also provided which enable the student to run his program one line at a time or to have otherwise invisible actions, such as assignment, dynamically commented. A similar facility for running a pre-specified number, possibly one, of commands was provided by Weyer and Cannara (1975) in their comparison of LOGO and SIMPER. SIMPER is a language to control a simulated simple decimal machine with an addressable store. Their debugging facilities did not have the sophistication of those provided by Barr et al. However they show that much can be done by generating appropriate messages at the user's teletype. They stress the need for error messages to be intelligible to the user, which means that they must reference the same virtual machine which the user knows about and not some other underlying machine. They also stress the value of program tracing facilities.

There are some dangers in providing complex facilities for observing the internal workings of the virtual machine. Control of these facilities may require the user to learn a further language, that is to control a second virtual machine which comments on the action of the first machine. Another danger is that the system with which the novice is interacting may start to look more and more

'intelligent'. The novice may question the reason why he is constrained to use a formal language which rejects poorly formed commands but which is still able to furnish what appear as helpful comments about its own workings or the possible causes of his errors.

Cannara (1976) gives examples of children attempting natural language conversation with both LOGO and SIMPER. Another error he noted was that children often formulated ambiguous commands and seemed to expect the computer to 'understand' what they meant. He also gives examples where children are clearly exploiting the non-intelligent responses of the machine for their own amusement.

The Story about the Virtual Machine

In addition to making the hidden actions of the virtual machine visible, novices may be helped by being given a description, and possibly a manipulatable model, of the virtual machine which is of sufficient precision to explain its action without being over-detailed. Construction of such representations is a difficult problem. Different users of the computer have different viewpoints on what is 'really' going on. For example, contrast the view of a system programmer with that of a maintenance engineer.

What the novice needs is a description which explains the events he sees. Thus if he is provided with a simple line editor which hides complex underlying file manipulations, such manipulations should not be part of the explanation of the editor. In the same way, the division of the Central Processor Unit into a Control Unit and an Arithmetic Unit need not be explained unless the novice needs the distinction in order to see how his virtual machine works. This point is argued by Barker (1975), who also argues that novices should

learn programming with a high-level language which allows interesting problems to be solved. In contrast, Cannara (1976) has some evidence that children learn programming well when they learn a high and a low level language simultaneously.

The advantage of providing novices with a concrete embodiment as well as a written description is argued by Mayer (1976). He gave an experimental group of students, learning a subset of FORTRAN, a board with labelled windows which they could use to construct and to hand trace through programs. The device had a pointer for marking the command currently being executed and erasable areas which modelled internal registers. The students were given a post-test in which they had to write programs to carry out various tasks. The performance of the experimental group of students was compared with a another group who were given no such model but received otherwise identical tutorial materials describing the action of the command set, and also with a third group who were given the model to use only during the post-test. Students who had learned programming using the simple model were better able to solve novel programming problems. Mayer scored as correct programs which contained syntactically incorrect statements. This was because the model was not of sufficient precision to account for the manner in which syntactic information would have been used to parse commands. It must be noted that the students did not interact with a 'real' computer, they had either to write programs or comprehend given programs.

Green et al. (1975), in describing their research into the design of conditionals, mention the problem of testing novice programmers who might well have not come across the idea of abstract algorithms, as we saw earlier. In order to find out which kind of

conditional formulation was comprehended most readily by novice programmers, Green et al. constructed an electro-mechanical device, "Lepus Experimentalis" and an associated story about a "hungry hare". The device was controlled by the simple programs written by the novices. In some senses the story the researchers used was far-fetched but it served to explain many of the actions of the small virtual machines they put at the the disposal of their subjects.

The way non-programmers organised commands, expressed in nearly natural English, into programs to sort cards under various constraints was studied by Miller (1974). He provided his subjects with a scenario, a story, and a physical model which were matched to the commands. The model consisted of boxes for the cards, and dishes for counters; in this way both data structures and program control could be modelled. The programming language was implemented on a computer. Subjects indicated which command they wished to assemble next into a program by typing a letter associated with that command. Thus there was no need to teach them about language syntax. He found that subjects were able to learn programming using these facilities and he examined the relative difficulty they had in programming different sorting problems.

The LOGO literature provides a number of examples of stories about the LOGO virtual machine, often based on anthropomorphic metaphors, though usually without the concrete models of say Mayer or Miller. For example, Statz (1973) describes how children played a game where each child took on the role of a LOGO primitive procedure and had to act accordingly. The "little man" metaphor of Feurzeig et al. (1969) has been widely used in different ways by LOGO researchers (see e.g. Brown and Rubinstein, 1973; Roman and Heller,

1975; Fischer, 1973). The programming curricula used by Weyer and Cannara (1975) and by Cannara (1976) have many stories and metaphors about the LOGO and SIMPER virtual machines. However some of the errors which their students made suggest that either the stories were not sufficiently well matched to the machine or that the students were confused by the variety of different virtual machines they had access to. Thus in SIMPER students confused editing and assembling and in LOGO they confused editing and executing. Only a small change of prompt indicated to the student that the virtual machine has changed. It would have been easy to overlook this or be eager to carry out the next task and forget to instruct the machine to change.

Models of the virtual machine and stories about its action depend on the underlying machine functioning correctly. Certain kinds of breakdown require explanations of the action of the computer at a much more detailed level than that given to the novice. However this need not necessarily be a handicap since the novice will usually be well aware that the story he has been told is a pedagogic device. As Green et al. (1975) note in relation to their story about the 'hungry hare', "subjects seem quite ready to swallow all this".

Simplicity and Consistency

As a result of his experiments with children (aged 9 to 15 years old), Cannara (1976) recommended changes to the implementations of LOGO and SIMPER which he used. He sought to make each language more self-consistent and simpler. Some of the changes to LOGO would have the secondary effect of making the language less like natural English. This is an important consideration as Green et al. (1975) point out. Thus Cannara recommended that 'noise' words be

eliminated. He suggested that the difference between the editing commands and the rest of LOGO should be reduced and that infix operators should be eliminated to make LOGO wholly prefix. Further he recommended that names chosen for LOGO primitives should accurately describe their action and not invoke an inappropriate connotation. There is a conflict here. On the one hand there is the desire to reduce the similarity of the formal programming language to natural English, to minimise the chance that the novice will attempt natural conversation with the machine. On the other hand one needs to use English words as names for primitive commands, and choose them carefully so that they evoke appropriate images of the action of the primitives. The latter consideration is mentioned by Kennedy (1975) who investigated how non-programmers in a hospital learned to use a computer-based patient-record system. The users had to create and update patients records and move records from one file to another, for instance if the patient changed doctor. One of the users' difficulties lay in their understanding of the terminology of 'records' and 'files' which had stricter definitions within the computer system than in everyday English. A further difficulty lay in the user choosing the wrong primitive to manipulate the data structure because the name of the primitive did not adequately indicate the action it performed. The user typed the instruction 'CHANGE' when changing a patient from one doctor to another. He should have typed the instruction 'TRANSFER' which transferred the patients record from the computer file of one doctor to another. 'CHANGE' was to be used to alter a patient's record, not transfer it. This difficulty can also be viewed as a result of not providing the user with a good representation of the virtual machine to which he

could relate the given commands. Indeed Kennedy uses the term "computer-naive" to describe precisely those users who lacked understanding of the computer and who consequently could not appreciate the effects of the commands they gave it.

Difficulties with LOGO

This subsection outlines the difficulties adult novice programmers are likely to face when learning LOGO. There are few sources of evidence regarding this problem. In the case of children the situation is a little better. The most detailed account of children's difficulties is that of Cannara (1976) which includes much of the data from an earlier experiment (Weyer and Cannara, 1975). Cannara taught children (aged from nine to fifteen) in two experiments designed to investigate how children learned programming. It is plausible that some of the difficulties he observed were caused by the immaturity of his subjects, although Austin (1976) and Statz (1973) suggest that adults make the same kind of mistakes as children when both are beginning to learn LOGO.

Some of the children's difficulties can be traced back to their misunderstanding of the underlying virtual machine. Canarra summarises their problem as follows:-

> "The two most common, virtually universal misunderstandings of all the students were: (1) misunderstandings of linguistic/computational context, and (2) ill-defined intents. The former applying to both storage/passing of information within their programs and their interaction with the interpreters. The latter, or fuzzy program specifications, amounts to wishful thinking, wherin the particular interpreter was expected to read

the student's mind and run correctly even though, for instance, a command had been left out." (p.109)

By 'linguistic/computational context', Cannara means the identity of the particular virtual machine with which the student is communicating. Thus some students attempted to run a program while it was being edited or vice versa. Difficulties with 'storage' included the belief that the computer stored partial results automatically and that these results could be retrieved for use in later computations. Students also had difficulties with the filing system.

There were also a number of difficulties associated with the name/value distinction. Some students thought that procedure names had to describe their action in order to work, or thought that variable names were computationally related to their values. Difficulties with control included misunderstandings about the recursion/iteration distinction, binding of argument values, result passing and command parsing. Syntactic difficulties included use of predicates, use of the abbreviation ':' for VALUE, use of infix operators and invention of illegal 'noise' words.

Cannara also notes that students were not very good at problem solving in programming. They did not break down problems into sub-problems nor did they match the problem decomposition to the available language constructs. They also failed to use old solutions in new problems and wrote inelegant procedures.

Statz (1973) briefly mentions some of the difficulties she observed among the eleven and twelve year old children in her experiment. There were wide variations among the children, caused by differences in the teaching curriculum as much as by differences in

ability or in maturation. She noted that some children had difficulty with filing and with editing. Others were defeated by "small mechanical errors" (later she mentions the difficulty some of the children had with spelling and typing). She also notes that children sometimes structured their procedures badly, as had children in Cannara's experiment. Milner (1973) in his experiment with eleven year old children mentions a difficulty they faced associated with the input of numerical values to procedures. A subsequent associated difficulty was the use of a variable as a counter to terminate a recursion and the definition of a procedure with two or more arguments.

The data on how adults learn LOGO are sparse. Austin (1976) reports that the student teachers he taught suffered from many of the same problems as the children in Cannara's experiment. He lists difficulties with editing, filing, breaking down problems appropriately and the construction of syntactically correct commands and procedures. Brown and Rubinstein (1974) report the social science students whom they taught also had difficulty in problem-solving. Some of the solutions to the problems set were so short that the students either saw the answer or not. There was no way for them to gradually refine an initial attempt. For other problems, students tended to use procedure 'templates' (e.g. a simple recursion) when they were not appropriate. Their students were confused by the scope of variables.

Statz mentions that some of the undergraduates she taught were "frightened away by initial problems" with the computer and that others "dealt with LOGO in a pedantic fashion". Particular programming difficulties were encountered in terminating a recursion

using a stop rule and understanding conditionals in general. Many also had difficulty with result passing. Others had formed the impression that 'input' (arguments) and 'output' (results) referred to user-typed input and computer-typed output. This is a good example of the wrong connotation being brought to bear because of the choice of name or term. Cannara also mentioned a similar difficulty and proposed a change in the terminolgy to reduce the difficulty. Statz also compared the programming performance of teachers (in a summer school) with that of the children taught in the main part of the Syracuse LOGO project (see section 2.2). She noted that they made the same kind of mistakes as children but were better able to recover from them. This is similar to Young's finding, above, in his comparision of naive and experienced programmers. The teachers, like Cannara's children, confused procedure execution with procedure definition, as well as making mistakes in the definition such as omitting line numbers. They also treated the machine as 'intelligent', for example, by answering error messages. Syntactic mistakes were also made with spaces and quote-signs. The causes of the last two errors probably lie in the form of the LOGO implementation used. There were command names constructed from concatenated English words such as IFTRUE, IFFALSE, TYPEIN, PENUP, PENDOWN which would not be interpreted if typed with a space inserted. The use of the quote-sign was inconsistent, especially in procedure declaration. This particular difficulty is discussed by Feurzeig (1977) in his draft specification for a more self-consistent LOGO implementation. Criticisms of LOGO are made by BRADY (1974). He argues that language implementation can have large effects on the user and that the type and form of the LOGO primitives should be

based on research rather than merely copying from its antecedent LISP, for example.

2.4 SUMMARY

Many student teachers perform badly in mathematics tests and have negative attitudes to mathematics even at the end of their courses in Colleges of Education. Little substantive research backs the claims for the value of learning mathematics through programming. There is limited evidence that 'rigour' and 'key concepts' are successfully introduced or that mathematical skills and attitudes to the subject are improved. There is only little evidence that problem-solving skills derived from programming are transferred. No research validates the argument that student teachers benefit, as teachers, from learning programming. The disadvantages of incorporating programming into a mathematics course have generally been minimised.

Learning programming is not as straightforward as proponents of of learning mathematics through programming suggest. There is evidence that novice programmers find some aspects of LOGO difficult and that adults make much the same mistakes as children. The main difficulties appear to be understanding the computational context, the name value distinction and problem-solving. A strategy for teaching programming based on a LOGO virtual machine has been argued.

CHAPTER 3

THE EXPERIMENT

This chapter gives an overview of the experiment conducted as part of this study. The first section states the problems tackled and describes the methods used in their investigation. The second section gives an overview of the experiment. The third section describes the programming language LOGO. The fourth and fifth sections outline the way in which programming and mathematics were taught.

3.1 METHODOLOGY

This thesis has investigated the problem of how student teachers might learn mathematics through computer programming. It has been concerned particularly with those students who have achieved low, formal mathematical qualifications, who dislike the subject and who are, consequently, most in need of mathematical help. It has investigated Papert's claim that it is "easy" to help such students by teaching them how to program in LOGO and by introducing them to mathematical concepts in a programming context.

Small numbers of students were recruited as volunteers. They undertook remedial mathematics work based on programming which supplemented the courses provided in their College of Education. The individual difficulties of the students determined the content of the remedial mathematics work. Individual case-studies provided data for the investigation of the following three sub-problems:-

(i) The mathematical difficulties of the students.

(ii) The difficulties arising out of teaching the students how

to program.

(iii) The effects of the programming experience on their understanding of mathematics.

Mathematical Difficulties

Studies using psychometric techniques have shown that many student teachers perform badly in written mathematics tests and have negative attitudes to the subject (see chapter 2). These studies indicate that a serious problem exists and that ways must be found to help such students. Without help they may be both unhappy and unsuccessful in teaching mathematics. Methods sensitive to students' individual difficulties are needed to determine appropriate remedial action. The present study employed two such methods.

(a) Observation of Teaching. The student teachers were observed and audio-recorded while they were on teaching practice. Attention was focused on the mathematical content of the lessons rather than on, say, teaching technique or classroom management. Particular attention was paid to the way the student explained mathematical ideas, to the tasks she set the children, and to the way she reacted to their difficulties and questions. Her poor explanations or her failures to provide the children with appropriate help were taken as evidence of possible mathematical difficulty on her part. Barnes (1973) provides an example of such an incident. He describes how a teacher, he was observing, rejected a child's correct illustration that three-sixths was equivalent to one-half (see figure 3.1).

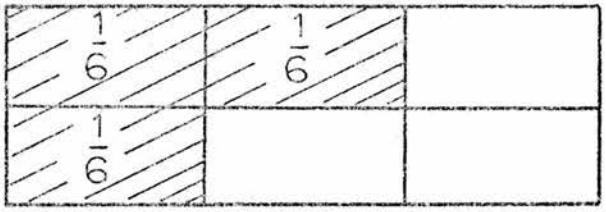


Figure 3.1. Three Sixths is Equivalent to One Half, from Barnes (1973).

In the present study such an incident was further investigated after the lesson by interviewing the student teacher.

(b) Interviews. The student teachers were interviewed about the lessons which had been observed. Selected extracts from the audio-recordings were played and the student's comments invited. Puzzling incidents (such as the example taken from Barnes, above) were investigated and formed the basis for discussion. These discussions were also audio-recorded. Ginsburg (1976) and Erlwanger (1973) have employed similar interview techniques to determine children's understanding of mathematics. In their hands, these techniques have proved sensitive enough to unravel complex mathematical difficulties which were as much linguistic as mathematical and might have passed unnoticed using psychometric techniques. The interviews were also used to probe the student teacher's attitude to mathematics and to teaching the subject. The student's comments about the lessons provided detail about which mathematical topics she liked or disliked, and on how she regarded her own mathematical competence.

The students chosen for this study were an atypical group. They

had low mathematical qualifications; most had only a pass at Scottish 'O' grade Arithmetic. They had a self-confessed dislike for the subject. They were also sufficiently worried (and conscientious) to spend considerable amounts of their free time attempting to remedy their mathematical difficulties. It was necessary to gauge the mathematical performance of these students against a wider sample. Rees' (1973) mathematics test was employed for this purpose and the performance of the students was compared to Rees' sample. The test also provided some evidence of topics in which the students performed poorly.

A questionnaire was given to the students which asked about their experience of mathematics as school pupils and about their present attitudes to the subject.

Programming Difficulties

Proponents of the mathematical value of learning to program have tended to underplay the difficulties which novices face in learning programming. Two sources of evidence of difficulty were used in the present study. First, a complete record of the students' dialogues with the computer were kept by means of 'dribble' files. These were files of everything which the student and the computer typed during a session. In addition, the students' calls for assistance and the ensuing tutorial conversations were audio-recorded. From these two sources of evidence it was possible to build up a picture of which aspects of programming the students found difficult. Questionnaires were also given to the students which provided evidence of the students' understanding of programming.

Effects of Programming

Having determined the mathematical difficulties of the students and having taught them the elements of LOGO programming, the author wrote mathematics worksheets designed for the students to explore their mathematical difficulties. The main source of evidence of the effects of the programming was the record of the students' mathematical work and the audio-recordings of conversations between the students and the author as they worked. A secondary source of evidence was the observation of lessons taught by the student. This could not be used as frequently as was desirable because the content of the students' lessons, while they were on teaching practice, was often prescribed by the school. Thus there were few opportunities to observe a lesson whose content was directly related to the work the student had undertaken through programming.

The questionnaires given to the students also provided evidence on their changing attitudes to mathematics. The comments of the students' tutors in the College of Education were also sought.

3.2 OVERVIEW OF THE EXPERIMENT

Two groups of students were recruited (see figure 3.2). Group 1 consisted of 8 students and joined the study in April 1975. Group 2 consisted of 7 students and joined the study in October 1975. All the students were taking a 3 year course leading to a Diploma of Education. The author visited the College of Education and gave a short presentation about the experiment. The students were told that they would be volunteers and that their work in the experiment would not be reported back to the College of Education.

GROUP 1

Anne	******	**	xxxxxx	******	****	******	
Betty	******	**	xxxxxx	******	**	******	
Fiona	******		xxxxxx	******	******		
Celia	******	**	xxxxxx	******	****		
Harry	******	**	xxxxxx	******			
Gail	****	**	xxxxxx				
Eve	****		xxxxxx				
Delia	****						

GROUP 2

Jane			******	xxxxxx		****	*x*x*x	******	**
Mary			****	xxxxxx	**	****	*x*x	******	
Irene			******	xxxxxx	****	**	*x*x*x		
Karen			****	xxxxxx	****				
Linda			**	xxxxxx	****				
Nina			**	xxxxxx	**				
Olive			****	xxxxxx					
	Summer Term	Summer Vac.	Winter Term	Spring Term	Summer Term	Summer Vac.	Winter Term	Spring Term	Summer Term
	A M J J A S O N D			J F M A M J J A S O N D				J F M A M J	
	1975			1976				1977	

** = month in which student programmed

xx = month in which student was on teaching practice

*x = month in which student programmed and was on teaching practice

Figure 3.2. Timescale of the Experiment.

Group 1

Eight students were recruited who were at the beginning of their final term in their second year. Details of the students are given in Appendix 1. This group acted as a pilot study and were used to try out early drafts of the programming and mathematics worksheets. Their major task during that term was to learn the elements of LOGO programming. Some of the students were sufficiently enthusiastic to continue working into their summer vacation. A number of meetings were arranged to take place within the College of Education. These were used for tutorial work to supplement the programming and also for testing purposes.

The winter term was devoted to teaching practice. Each of the students, except Delia who had dropped out earlier, was visited twice and a mathematics lesson was audio-recorded. Each student then had the opportunity to listen to the recording with the tutor and comment on it. These sessions were not recorded, though notes were made. During this term the programming language implementation was transferred to a different computer with a small number of changes. This implemenation was an interim measure and was rather unreliable. These changes are described in section 3.4 of this chapter.

In the spring term, the students spent some time revising programming after the gap caused by teaching practice. Then they tackled a number of mathematical projects in programming. Again meetings were arranged in the College of Education where a number of tests and questionnaires were tried out for later use with group 2. Some tutorial work was also undertaken to supplement the programming.

By the time the summer term started only four of the original

eight students were still taking part in the study. All who had dropped out gave their reasons as pressure of work within the College of Education and impending Diploma examinations.

Again the programming language implementation was transferred to a new computer with a number of detail and syntactic changes in the language. At this time both the interim and this final implementation were available although neither was particularly reliable. The students worked with both, often swopping after a crash in one implementation, and became adept at coping with the language variations. This was a most unfortunate state of affairs and poor pedagogic practice. However as the students had to make a special journey in their own time to the University it seemed preferable that they had some chance to program rather than none. This term further mathematical projects were undertaken and a number of tutorial sessions held to discuss links between programming and mathematics. The students' work this term was limited by their Diploma examinations and disrupted by a sit-in within the College of Education to protest at the bleak job opportunities for newly qualified students.

At the end of the term the students graduated with Diplomas in Education though with little prospect of employment. Two students returned to the College to take a further teaching qualification. These two students continued to program during the succeeding summer vacation.

Group 2

Group 2 was recruited at the beginning of the winter term 1975. Details of the students are given in Appendix 1. Some difficulty was

experienced because some students agreed to join the study but then had second thoughts. Other students had to be recruited in their place. This explains the uneven start to the group's work. They started to learn the elements of programming using the interim and rather unreliable language implementation, mentioned in the last section. Within the College of Education meetings were arranged at which the students took part in a discussion and answered a questionnaire about their own experience of mathematics as school pupils. This was the first of three questionnaires given to this group. It will be referred to in future as questionnaire (1) and is given in Appendix 2.

The students also took the Rees (1973) mathematics test, described in chapter 4. This was to compare this group with Rees' larger sample of student teachers. Two tutorials were held to discuss some of the mathematical difficulties which had been noted in their answers to the test. One tutorial attempted to be a 'Socratic Dialogue' on how the rule about dividing fractions is derived; this was something many of the students wished to know. The author judged this tutorial as unsuccessful because the students were not given sufficient opportunity to explore the problem for themselves. The students were not very forthcoming and what had intended to be a dialogue turned into a monologue. The failure of the 'dialogue' is further emphasised by the fact that Jane asked about the derivation of the fraction division rule later in the year.

At the end of the term arrangements were made to video-tape each of the students teaching fractions to two children under laboratory conditions. This was to provide data for that part of the study which had originally intended to investigate how teaching style was

affected by the programming experience (du Boulay, 1977). Although these recordings were made, this part of the study was discontinued because the scope of the study was too broad for a single person to undertake. Also at that time the links between teaching children in classrooms and learning programming were insufficiently developed for incorporation into the work undertaken by the students.

The next term, in the spring of 1976, the students were all on teaching practice. Each student was visited twice and a mathematics lesson audio-recorded. In most cases the student came later to the university where she heard and commented on the recording. These discussions were also audio-recorded.

Apart from providing data on the students' mathematical difficulties, the observation of lessons served a number of other purposes, as it had done for students in group 1. It kept the author in touch with the students during what otherwise would have been a long gap in their association with the study. It showed that the author was concerned about the difficulties they were facing in the classroom and that the work they undertook in programming sessions would address these difficulties. It was designed to build an atmosphere of trust between the student and the tutor so that mathematical difficulties could be brought out into the open.

The discussions about the lessons were designed to elicit the student's views of the lesson and to clarify incidents which had taken place. It proved impossible and undesirable for the tutor to remain a neutral observer in these discussions since the students would ask for practical advice about classroom management or about the mathematical content of their lessons. These sessions took on much of the flavour of what Eggleston et al. (1975) have called

'prescriptive' studies of classroom interaction. That is, advice was given to the students based on the tutor's educational prejudices and practice.

When the students returned to programming in the summer term they had to use the third programming language implementation. This had a number of changes compared to the second implementation in which they had originally learned programming. Thus much of their work was devoted to relearning the language because of these changes, the long gap since they had last programmed and the slow start the previous term. Only a small amount of work of obvious and direct mathematical relevance was undertaken. Only a single meeting was held in the College of Education because of the disruption caused by the sit-in mentioned in the last section. At this meeting students filled out a questionnaire to test their understanding of programming and to find out their views of the effects of their programming experience to date. This questionnaire will be referred to in future as questionnaire (2) and is given in Appendix 3.

Despite this rather unpromising start, three students opted to continue with the study for another year. These three were Jane, Irene and Mary. Case studies of their work will be presented in chapters 6 and 7. All three continued to program through part of their summer vacation. Some of this work involved mathematical projects.

In the winter term these three students had teaching practice again, as they were now in their third and final year of Diploma studies. Each student was visited at approximately weekly intervals and recordings made of mathematics lessons. Jane was observed 7 times, Mary 7 times and Irene 5 times. As before they listened to

and commented on these recordings and these discussions were also recorded. In addition they attended programming sessions during teaching practice. Often these two activities were combined, the student first hearing a recording of a lesson and then doing some programming. This term Jane and Mary often came together, as they had during the preceding vacation, and listened to each other's lessons. They had become friends and used these sessions as a good opportunity to swop news, since they had been posted to different schools. Having two students at the discussion of the lessons was beneficial because it changed the dynamics of the situation. The tutor could take a more background position while the two students discussed the incidents in the classroom.

Most of the programming work took the form of mathematical projects. Some of these were derived from difficulties observed in the classroom. Others stemmed from difficulties reported by the student but not observed by the author. At the end of this term a third and final questionnaire was sent out to all the students, including those in group 1, asking for their views on their experience of programming. This questionnaire will be referred to in future as questionnaire (3) and is given in Appendix 4. The College of Education Tutors of Jane, Mary and Irene were also approached for their views of these students' abilities as teachers of mathematics.

The mathematical work continued in the Spring of 1977, though not by Irene who had decided that the impending examinations and increasing commitments forced her to stop. This term some programming work was also undertaken to help sort out difficulties experienced by Jane and Mary in the mathematics course they were taking in the College of Education. Because of their mathematical

weakness they had decided to study mathematics as one of their principal subjects for their Diploma. But they had difficulty understanding some of the topics in the lectures, for example matrix operations and groups of transformations.

Jane and Mary stopped programming work during the early part of the summer term to concentrate on their Diploma examination revision. Once the examinations were over Jane returned for several sessions and undertook programming work on fraction and decimal manipulations.

3.3 LOGO

Before describing how LOGO was taught, it is necessary to outline the properties of this language. We describe the final implementation of LOGO used in this study. A complete description of a very similar implementation is given by McArthur (1974). LOGO was first described in detail by Feurzeig et al. (1969). The language is derived from LISP (McCarthy, 1969) with which it shares many features. In the following description some terms are given in single quotes, e.g. 'working memory'. These are the terms used in the teaching materials.

Data-types and Primitives

LOGO has three data-types: integers, words and lists. Integers consist of strings of numeric digits. Words consist of strings of alphanumeric characters. A single quote in front of a word denotes that it is a literal:-

"FRED

Lists are ordered sequences of integers, words or sub-lists which may be nested to any depth. Lists are delimited by square brackets e.g.

[6 [FRED SMITH] 47 C]

The uniformity of data and program representation, which is the main feature of LISP, is lost in this implementation of LOGO (at least as far as the user is concerned).

Primitives fall into three classes. In the first, a primitive is executed because it produces some side-effect in either the internal environment of the machine or in the external environment of peripheral devices. The second class contains primitives, corresponding to LISP functions, that return a value, usually by performing some transformation on data. This value will be either an integer, a word or a list. A third class contains hybrid primitives which return a value and produce a side effect. These correspond to what McCarthy calls "pseudo-functions" in LISP.

The basic activity of the LOGO programmer is the definition of procedures. A procedure is a numbered sequence of commands which is given a unique name by the user. User-defined procedures fall into the same three classes as the primitives and may be used interchangeably with them. The user is thus able to extend the instruction set. User-defined procedures may call other user-defined procedures as sub-procedures. Recursive definitions are allowed.

A command consists of a call to a primitive or to a user-defined procedure. In future both primitives and user-defined procedures will be referred to simply as procedures. In general a procedure will require an argument value. This value will be a word, integer or list and will be supplied by a literal, by returning the value associated with a variable name or by evaluating an expression. An expression consists of a call to a procedure which itself may require argument values. A command may consist of a composition of procedure calls which may be nested to any depth.

When the system is 'waiting' for a command it issues the prompt 'W:'. An example of a command is given below which has the effect of moving the 'turtle' forward 100 units.

W: FORWARD MULTIPLY 5 ADD 16 4

To improve readability, expressions may be delimited with parentheses, as in the following command which has an identical effect:-

W: FORWARD (MULTIPLY 5 (ADD 16 4))

The interpreter reads a command from the user's console and then evaluates and executes it. Each command must produce some side effect and may not simply compute a value. Commands continue to be read until the user gives the command GOODBYE which has the effect of terminating the session.

User-defined Procedures

User-defined procedures either return a value, produce a side-effect, or in some cases do both. Side effects fall into two categories: those in the external environment of peripheral devices and those in the internal environment of the user's workspace ('working memory') or his long term file storage ('permanent memory').

There is a range of output devices which include: a small floor robot with an attached pen known as a 'turtle', two graph-plotters with their associated 'compasses', a storage tube display, and a single voice tone generator. The graph-plotters and storage tube display accept the same commands as the turtle and simulate its action, but produce rather faster and more accurate drawings. The compasses attached to the graph-plotters indicate the current heading

of the simulated turtle (Kemplay and du Boulay, 1976).

The user normally inputs commands via a teletype. She may also construct simple procedures using a special input device known as a 'button-box' (du Boulay and Emanuel, 1975). This has sixteen buttons covered with a transparent overlay on which the button labels are written. The labels consist of commands for the turtle, see figure 3.3.

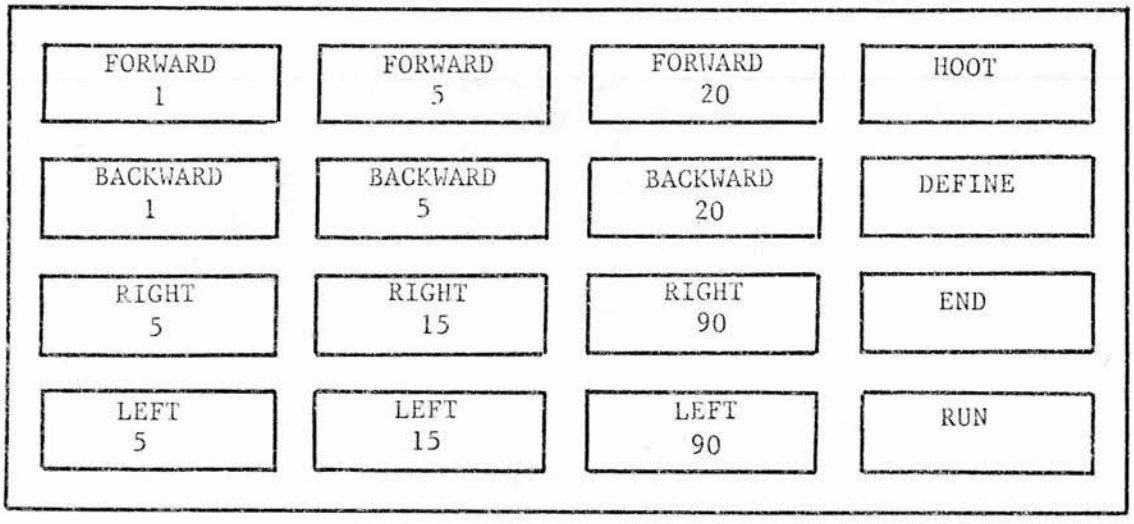


Figure 3.3. The Button Box.

Each button lights up while the turtle is executing its command. The user can store a sequence of commands by pressing the button labelled DEFINE. All further commands are stored and not executed until the button marked END is pressed. By pressing the button marked RUN, the stored sequence of commands (a procedure) will be executed. Flow of control through the sequence is indicated by the buttons which light up in turn as the command associated with each

button is executed. A different overlay and program enables the button-box to be used to control the tone-generator. In this case, stored procedures correspond to tunes rather than drawings.

Using a teletype, the user can define a procedure by typing DEFINE in response to the prompt 'W:'. This primitive takes a variable number of arguments. The first argument becomes the name of the new procedure. Succeeding arguments are the names of the formal parameters ('input names') of the new procedure. All names must be quoted words. If this command is successfully executed the machine is put into the 'defining state' and the prompt changes to an indented 'D:'. The user then types in the commands (in any order) which constitute her procedure. Each command must be preceded by a line number which defines the subsequent order of execution of the commands. The following definition is of a procedure named VEE with a single formal parameter named SIZE.

```
W: DEFINE "VEE "SIZE
    D: 10 FORWARD VALUE "SIZE
    D: 20 BACKWARD VALUE "SIZE
    D: 30 LEFT 45
    D: 40 FORWARD VALUE "SIZE
    D: END
```

The procedure can be run by typing its unquoted name and by supplying a suitable argument value ('input value'), e.g. :-

```
W: VEE 83
```

This will draw the shape in figure 3.4.

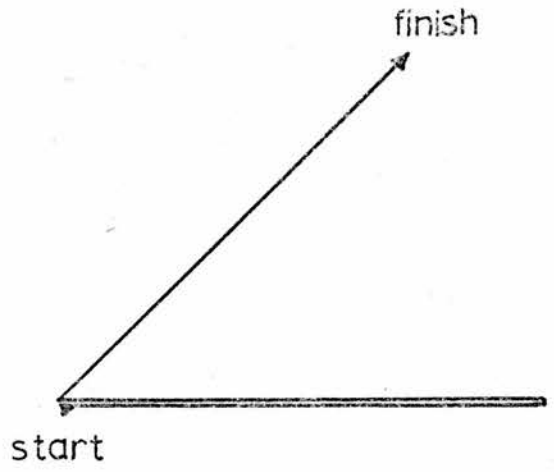


Figure 3.4. Drawing a VEE.

This procedure is not state-transparent: it leaves the turtle in a different position and with a different heading compared to its initial state.

Internal side-effects consist of such changes as procedure definition and editing, assignment, copying a procedure between working and permanent memory, and disrupting flow of control through a procedure. Procedure definition has already been described. There is a simple line editor. It is entered by running the primitive CHANGE with a single argument which is the quoted name of the procedure to be edited. Commands are typed as for procedure definition. Any command already existing with the same line number is overwritten. Exit from the editor is achieved by typing END. Both defining and editing can only accept input from the user's console and so the dynamic creation and modification of procedures is not possible.

Individual procedures may be copied from the user's workspace to his permanent file store or vice versa. Existing procedures of the same name are overwritten. Thus as far as the user is concerned the monitor system is written in LOGO. The initial login sequence is

short and thereafter, apart from breakdowns, the underlying system is hidden from the user.

Assignment is accomplished using the primitive MAKE. It takes two arguments which are both evaluated. The first, which must evaluate to a quoted word, will be the name for the value of the second argument. At command level, MAKE creates or updates global variables. When called within a user procedure, MAKE first checks whether the procedure has formal or local parameters with the given name. If so, that variable is updated. If not, outstanding procedures are examined, most recent first, to see if any of them have either a formal or local parameter of the given name. If so that variable is updated. Only if this search is unsuccessful is a global created or updated.

Flow of control through a user defined procedure can be changed in various ways. There is a primitive REPEAT <integer> <command> which controls iteration. There are conditionals in two forms. One form is IF <predicate> THEN <command> or IF <predicate> THEN <command> ELSE <command>. The other form sets a flag which can be subsequently examined. The flag is set by TEST <predicate> and is acted upon by either IFTRUE <command> or IFFALSE <command>. Control may be directed to a line other than the subsequent one by using the primitive GO <line-number>. Immediate termination of a procedure is accomplished with the primitive STOP. A user-defined procedure can request information to be typed at the console by employing the primitive REPLY. This temporarily suspends execution of the procedure in which it is embedded and waits for the user to type something at his console. Whatever is typed is unevaluated and made into a list. This list is returned as the result of REPLY. This

primitive, therefore, falls into that third class of primitives which both return a value and produce a side-effect.

Procedures which return a value may also be defined by the user. Like primitives, these may or may not have parameters. Such a procedure must exit via a call to the primitive RESULT. This primitive evaluates its argument and terminates execution of the user-procedure in which it is called. The argument value becomes the returned value of the user-defined procedure. The user must ensure that the value returned by a procedure is used as the argument for another procedure. No command is allowed to return a value, so the following command is illegal because the computed value, 9, is not used:-

```
W: ADD 4 5
```

Such a command produces no overall change in the system and is effectively a null command. The execution of primitive and user-defined procedures may be traced. Procedure entry, argument values and any computed result are dynamically commented on the console. The following table (figure 3.5) gives brief details of sixteen commonly used LOGO primitives.

PRIMITIVES	ARGUMENTS	RETURNED VALUE	SIDE-EFFECT
FORWARD	integer	none	Moves turtle forward in current direction.
BACKWARD	integer	none	Moves turtle backward in current direction.
LEFT	integer	none	Rotates turtle on spot anticlockwise.
RIGHT	integer	none	Rotates turtle on spot clockwise.
PRINT	integer/word/list	none	Prints argument at console.
DEFINE	word(s)	none	Puts system in define state.
CHANGE	word	none	Puts system in define state for editing.
END	none	none	Returns system to waiting state.
REMEMBER	word/list	none	Stores procedure(s) in permanent file.
RECALL	word/list	none	Retrieves procedure(s) from permanent file.
SHOW	word	none	Lists user-defined procedure.
RESULT	integer/word/list	same as argument	Returns value of argument as value of procedure in which it is embedded.
REPEAT	integer+command	same as command	Repeats execution of command.
VALUE	word	integer/ word/list	Retrieves value associated with name given as argument.
ADD	integer+integer	integer	None, returns sum of arguments.
MULTIPLY	integer+integer	integer	None, returns product of arguments.

Figure 3.5. Commonly Used LOGO Primitives

3.4 TEACHING PROGRAMMING

The original objective was to teach the students the elements of LOGO programming including variables, iteration, recursion and conditionals. This was seen as an essential prerequisite to the mathematical programming work which it was intended that the students carry out. It was also intended to teach those problem solving strategies which are normally associated with LOGO programming e.g. problem decomposition by sub-procedurisation.

All the students were taught programming in a room equipped with four teletypes and four drawing devices (a turtle, a storage-tube display and two graph-plotters) and the button-box. In the room there were also items to be found in a classroom, e.g. a blackboard, paper and pencils etc. Students attended the programming sessions in groups of four or less. The programming facilities used in this thesis were initially set up by a research project to establish how children's understanding of school mathematics was affected by learning LOGO programming (Howe and O'Shea, 1976). The children's research project ran concurrently with the research reported here. Three different implementations of LOGO were used in the present study. Although the implementation changes improved the language, their introduction in the middle of the student teachers' work was disruptive but unavoidable. The direction of the changes which occurred in the implementations were as a result both of the experience of the student teachers and the children in the other research project.

Three Implementations of LOGO

The language changes followed the strategy argued in chapter 2. The changes were introduced to reveal the workings of the virtual machine, to make the virtual machine simpler or more self-consistent, or to provide a more uniform set of descriptive terms for the actions and parts of the virtual machine. This produced changes in the names of the primitives, some changes of syntax, and a complete re-write of the error messages to make them more understandable. The following chart (figure 3.6) details the main differences between the first and third implementation which the user could detect. The second implementation was very similar to the first and involved merely a change of host machine. One of the few noticeable changes which this produced was an increase in the stack depth of nested procedure calls. The first and second implementations were written in POP-2 (Burstall et al., 1971) for an ICL4130 and a PDP10 respectively by McArthur (1973). The final implementation was written in IMP (ERCC, 1974) for an ICL 4-75 by McArthur.

1st IMPLEM.	3rd IMPLEM	COMMENT
TO FOO	DEFINE "FOO	Name and syntax of procedure definition changed. Non-evaluation of procedure name marked syntactically. Procedure cannot be erased by accidental redefinition.
TO FOO :INP	DEFINE "FOO "INP	Formal parameter names given as quoted words.
10 FORWARD :INP	10 FORWARD VALUE "INP	Parameter value retrieved by running VALUE.
SHOW FOO	SHOW "FOO	Quote used consistently to inhibit execution of procedures.
SUM EDIT PENUP	ADD CHANGE LIFT	Many names changed to more familiar terms. Concatenated and double word procedure names eliminated.
1: &:	W: D:	Prompts match description of virtual machine.
2:	W:	System quits to command level on error, does not recursively reenter LOGO system.
All procedures return a value	Selected procedures return a value	Distinction between returning a value and producing a side effect emphasised.
Error messages		Much more explicit, refer to same virtual machine as in teaching notes.
Tracing		Better tracing facilities implemented.

Figure 3.6. Changes in LOGO.

The following chart, figure 3.7, illustrates which implementation was used by each group of students.

GROUP 1

FIRST SECOND THIRD	111111	11		222222	222222 333333	333333	

GROUP 2

SECOND THIRD			222222		3333	3333	333333	333333	33
	Summer Term	Summer Vac.	Winter Term	Spring Term	Summer Term	Summer Vac.	Winter Term	Spring Term	Summer Term
	A M J J A S O N D 1975			J F M A M J J A S O N D 1976				J F M A M J 1977	

11 = First Implementation
22 = Second Implementation
33 = Third Implementation

Figure 3.7. Implementations Used by Each Group.

Group 1

The first group of students (group 1) was taught programming using a series of notes which described different programming topics and a separate series of worksheets which presented exercises and problems. The students generally worked through the materials at their consoles. The tutor usually described the objective of each note or worksheet to the student as it was issued. There were also a number of tutorial sessions on programming held in the College of Education which were attended by the whole group.

Initially only descriptive notes on LOGO were issued and students were expected to formulate their own exercises and problems from the examples in the notes. However, many of the students

disliked this approach and welcomed the first worksheet when it was issued, since it gave precise instructions as to what they should do. Accordingly, further worksheets were issued in parallel with the teaching notes. The following table (figure 3.8) outlines the content of the 12 teaching notes and 7 worksheets issued to the group during the period April to July, 1975.

NOTE	WORKSHEET	CONTENT
1		BUTTON-BOX: Turtle, drawing, commands, procedures, running, defining.
2		DRAWING DEVICES: Logging in, teletype, drawing, logging out.
3		PROCEDURES: Defining, editing, storing and retrieving procedures.
4		COMPUTER: Two memories, CPU, hardware layout.
5		SUB-PROCEDURES: Procedures and sub-procedures, REPEAT, problem decomposition.
6		WORDS, LISTS: Printing numbers, words and lists.
	1	Manipulating data, predicates.
7		RESULTS, EFFECTS: Distinction between side-effects and returning values in primitives. Parsing prefix commands.
8		INPUTS: User-procedures with parameters.
	2	Procedures taking inputs, examples from drawing and symbol manipulation.
9		RUNNING PROCEDURES: Little-man analogy for procedure calls.
	3	Hierarchy of procedures with inputs.
10		CONDITIONALS: TEST, IFTRUE, IFFALSE. Communication between user-procedure and console, GETWORD, GETLIST.
	4	Use of conditionals and predicates. Quiz procedures.
11		RECURSION: Middle line recursion.
	5	Recursive drawing procedures, no stop rule.
	6	Recursive drawing procedures with stop rule.
12		OUTPUT RESULT: Defining procedures which return a value.
	7	Mini projects.

Figure 3.8. Notes and Worksheets for Group 1.

In January 1976 the students returned to programming after teaching practice. LOGO had been transferred to a new host machine (implementation 2). This implementation retained all the faults of the old implementation, e.g. poor error messages, and was also much more unreliable. Fifteen out of the twenty-one sessions held that term were disrupted by computer failures or unexpected operator interventions.

The third implementation of LOGO was being developed at that time. It was decided to use this implementation when the second implementation was not available. This meant difficulties for the students because of the differences between the two implementations and because the students could not communicate their stored procedures between the two host machines. Unfortunately the third implementation of LOGO had teething troubles and was also rather unreliable.

The students were issued with 4 revision worksheets which suggested a number of programming problems using the LOGO constructs they had learned earlier. The students were also introduced to the 'IF <predicate> THEN <command>' form of conditional, since 'TEST, IFTRUE, IFFALSE' had caused difficulty. It had been badly explained and had a bug in its implementation. The students were also given a note describing the differences between the implementations of LOGO.

In the summer term, 1976, it was decided to use only the third implementation of LOGO. Its reliability had improved and only a few sessions were disrupted. The four students still attending the sessions worked on a variety of programming projects, some of their own choosing and others devised for them. No new notes on LOGO were issued.

Group 2

The second group of students (group 2) started programming one term later than group 1. A new set of teaching notes were written which incorporated changes due to the experience with group 1. Teaching notes and worksheets were combined and there were alterations in presentation and in the order of topics. The objectives were as before: to teach the elements of LOGO and problem-solving stategies. The introductory sessions were disrupted by computer failures, but later sessions were more reliable.

The content of the notes issued are set out in the following table (see figure 3.9).

Group 2 had fewer programming sessions in their first term of programming and covered less ground than group 1. When group 2 returned to programming after teaching practice, in the summer term of 1976, they had to learn how to use the third implementation of LOGO. The students were given a series of revision notes which contained information and exercises and explained the implementation changes. By this time, a more comprehensive programming primer was being completed in conjunction with the childrens' research project (du Boulay and O'Shea, 1976).

NO.	CONTENT
1	BUTTON BOX: Commands, effects, memory, storing procedure, turtle, music box (single voice tone generator).
2	DRAWING DEVICES: Teletype, drawing procedures,login,logout, Typing errors.
3	ERROR MESSAGES: Common error messages.
4	PROCEDURES: Defining procedures.
5	ITERATION: Using REPEAT.
6	EDITING: Changing and listing a procedure.
7	MEMORY: Working and permanent memory, copying procedures between memories.
8	CALCULATOR: Simple calculations in LOGO. Using PRINT, ADD etc.
9	RESULTS,EFFECTS: Distinction between returning a value and producing a side effect.
10	PARSING: Parsing prefix commands.
11	LITTLE MEN: Little men diagrams as analogy for procedure calls.
12	INPUTS: Procedures with parameters.
13	PROBLEM DECOMPOSITION: Procedures and sub-procedures.

Figure 3.9. Worksheets for Group 2.

During the latter half of the summer term, in the summer vacation and for the next two terms, notes from this extended primer were used. The topics covered in this period are set out in the table (see figure 3.10) below. The numbers refer to chapters of the primer.

NO.	CONTENT
9	SIMPLE CALCULATIONS: Using PRINT, ADD to form simple prefix commands.
10	PARSING: Parsing prefix commands.
12	PROBLEM DECOMPOSITION: Procedures and sub-procedures.
13	INPUTS: Procedures with parameters.
14	INPUTS: Sub-procedures with parameters.
15	EDITING: Further editing primitives.
16	TWO MEMORIES: Further primitives for storing and retrieving procedures.
17	INPUTS: Super-procedure passing argument value to sub-procedure.
18	POLYGONS: Drawing polygons using iteration.
19	RESULTS: User-defined procedures which return a value.
20	RECURSION: Recursion without stop rules. Changing context.
21	SPIRALS: Last line recursion drawing spirals.
22	PREDICATES: Primitive predicates.
23	CONDITIONALS: IF...THEN...ELSE.
24	QUIZES: User-defined procedure communicating with console.

Figure 3.10. Primer Notes for Group 2.

Primer chapters on assignment and GO (goto) were shown to one student (Mary) for her work on flowcharts. She was also introduced to RANDOM for work on probability.

Students in group 2 did not progress far enough to write recursive procedures with stop rules, whereas students in group 1 did write such procedures. Neither group of students wrote procedures involving constructive recursion or list processing (except trivial cases).

3.5 TEACHING MATHEMATICS

The following chart (see figure 3.11) outlines the mathematics worksheets which were issued (mainly to group 2).

TOPIC/TITLE	CONTENTS	TYPICAL TASK
Counting	Matching number names to objects.	Write procedures to mimic different stages of counting ability.
Multiplication and division.	Multiplication as repeated addition, division as partition or quotition.	Write procedures to manipulate lists in ways which illustrate multiplication and division.
Fractions	Fractions as a double operation of multiplication and division.	Write procedures to add fractions expressed as two element lists.
Fractions	Part/whole relation.	Write procedures to draw fractional parts of a disc.
Fractions	Addition of fractions.	Write procedures to put parts of discs together.
Fractions	Ratios, multiplication of ratios.	Run given procedure which draws shapes in given ratio.
Fractions	Ratios, division of ratios.	Run given procedure to illustrate division of ratios.
Decimals	Multiplication of decimal fractions.	Run given procedures which illustrate partial products in multiplication process.
Directed numbers.	Four operations on directed numbers.	Use FORWARD and BACKWARD to illustrate directed numbers and operations on them.

Figure 3.11. Mathematics Worksheets

(continued on next page)

(continued)

TOPIC/TITLE	CONTENTS	TYPICAL TASK
Vectors	Geometric vectors.	Illustrate geometric vectors using drawing commands.
Positioning the turtle.	Coordinates.	Write procedure to position the turtle anywhere in its drawing area.
Triangles	Classes of triangle, interior/exterior angles.	Match classes of triangle to given procedures, Investigate angle properties of polygons.
Patterns	Translations, rotations and reflections.	Write procedures which repeatedly translate or rotate a shape, investigate effect of swopping LEFT and RIGHT or FORWARD and BACKWARD.
Symmetry	Group of transformations of the rectangle.	Run given procedures which isometrically transform rectangle, fill in group table.
Length, Area Volume.	Measurement, tessellations.	Write procedures which fill given space with regular polygons.
Probability	Random walks, dice.	Write procedure to drive turtle on random path, write procedure to act as dice, investigate effects of two dice together.
Flowcharts	Simple flowcharts	Draw flowcharts to illustrate LOGO procedures.
Flowcharts	Assignment, goto.	Write procedures which illustrate given flowcharts, special flowchart conventions.

Figure 3.11. Mathematics Worksheets.
(continued from previous page)

CHAPTER 4

MATHEMATICAL DIFFICULTIES

Two broad types of mathematical difficulty emerged from observation of the student teachers. The first concerned the distinction between knowing and understanding a mathematical concept. This was characterised by the student's inability to explain or justify mathematical rules which she knew how to apply e.g. the classic 'turn upside-down and multiply' rule for dividing by a fraction. The second difficulty concerned the student's inability to express mathematical ideas in English. This was characterised by the imprecise and muddled explanations which the students offered their pupils. For example one student mixed up two different illustrations for division in a most confusing manner. Further the students showed themselves to be ignorant of certain topics, e.g. integer operations, and they also performed poorly in a mathematics test. The students were well aware of their low mathematical competence. This made them anxious and lowered their self-confidence. So much so, that they even mistrusted their ability in topics which they did understand.

Previous studies, using written tests, have found mathematics topics in which students performed poorly (e.g. the 'common core' of Rees, 1973). The inference to be drawn from such studies is that the topics in question will be taught badly. Here we show that the converse - topics in which the student performs well, she will teach well - is not true. The two types of difficulty mentioned above, 'no explanation' and 'muddled explanation', were found for topics in which the student could answer written questions correctly. This

shows that courses for student teachers must pay particular attention to the difficulties which students face in explaining mathematics. Designers of these courses cannot assume that their students are well equipped to teach the topics on which they have passed a written examination.

4.1 INABILITY TO EXPLAIN RULES

Students frequently knew the mathematical rules for some topic but could offer little in the way of explanation, justification or meaning for the rules. Irene blamed the way she had been taught at school and she was in danger of repeating the mistakes of those who had taught her.

> "I could never see any real value in it [mathematics]. I was never given reasons why such and such was so."

Students would often be at a loss for a justification of the algorithms which they taught their pupils. This made the students anxious. Firstly because they feared that a pupil might ask them why the rules were the way they had described. Secondly because this method of teaching was at odds with their ideals. They wanted to teach 'for understanding' but sometimes had only the mathematical resources to promote 'rote-learning'.

Jane complained that she did not know why the rule for positioning the decimal point after multiplication (or division) of decimal numbers worked. In multiplication one adds the number of decimal places of the multiplier to that of the multiplicand to find the number of decimal places in the product. In discussion she revealed how sparse her knowledge of the decimal numeration system

was. She found it very difficult to multiply or divide by powers of ten by 'shifting the point' or by 'adding or subtracting zeros'.

Jane: "You see we had to do it. We had to do that sort of thing at [the College of Education]."

Author: "Uhu."

Jane: "But they shouldn't really teach us exactly because I always had to do it the long way...you know, a thousand into so and so...you know, she expected us to do it mentally, but I could never."

Author: "You mean shifting."

Jane: "Yes, shifting it...but I could never do it mentally. I always had to write it down because I could never see how you can, you know, why or how you could shift the point in a way that you could remember."

Jane had similar difficulties with fractions and percentages. She explained that she did not understand enough to tackle "anything to do with percentages". She had been asked by the class teacher to help some children with this topic, and she reported:

"I didn't really feel confident enough to really show them in case I was doing it the wrong way from Mrs. L...."

Jane worried that although she knew the rule for dividing fractions she had no idea why the rule worked or how she might illustrate its meaning.

Difficulties with fractions were common among the students (and have been reported in the literature, see e.g. Lumb, 1974). Betty and Mary were confused by the apparently bizarre way in which fractions behaved compared to natural numbers, i.e. when they divided by a fraction their answer could be bigger than the number

they started with:

$$12/(1/2) = 24$$

Fiona had much the same difficulty, and an extract from a lesson of hers where this was apparent will be given. Betty felt that the course in the College of Education was not providing her with the kind of explanation she needed and expected. Betty's low opinion of herself is evident in the following conversation.

Betty: "The difference is that I am not afraid to talk to you, to tell you how stupid I am"

Author: "But, aye, you aren't stupid."

Betty: "The people at [the College of Education], I mean, well I got to the stage where I just wouldn't talk to them. Because I felt they knew the answer and for me to come out and say, well, 'why should that work? I just can't understand that'. I mean just like when I was at school, it took me ages, you know, when you were dividing a number, fraction, em, how it got, em, a bigger number. As far as I was concerned, you know, three twelvths."

Author: "Yes."

Betty: "If you were dividing it, I couldn't work that out."

Author: "Do you mean to say, if you took what, er, twenty four divided by three twelvths, how you got a number bigger than twenty-four?"

Betty: "Yes."

The extract shows how Betty worried about not understanding those elementary parts of mathematics which were taken for granted in her course. Betty found the dynamic interpretation of an angle as a rotation hard to understand compared to her static interpretation of

it as merely "two wee lines".

"Yes but to me it didn't relate as a turn."

"...I could never really fit it into three sixty in my mind, even though I knew that angles made up three sixty. It didn't really mean anything."

"...and why it should talk about a revolution."

Jane admitted that although she knew some of the formulae for calculating area and volume, she did not understand why the formulae worked. She said that she did not know the area formulae for circles and triangles.

Using a Compass

Both Jane and Nina did not understand how to use certain geometric instruments. Nina was teaching her class how to construct isosceles triangles, given the lengths of their sides. She made the following construction on the blackboard. First she drew a line AB of given length. Then she opened the compass to the second given length and with centres A and B drew two arcs long enough to intersect with the line AB (see figure 4.1). She then joined the point, where the arcs intersected, to the points where the arcs cut AB, instead of to A and to B.

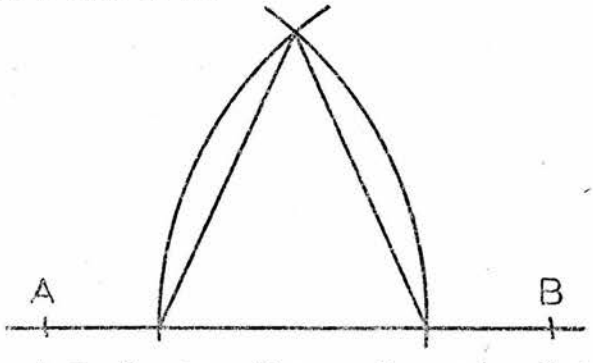


Figure 4.1. Constructing an Isosceles Triangle.

Jane discussed a lesson she had given on the construction of different classes of triangle. She pointed out that it was only in the College of Education that she had understood the function of the compass arcs in constructions. Unfortunately she did not teach this concept to the children. The lesson consisted of her telling the children a number of ruler, compass and protractor construction algorithms. Each construction was linked to a triangle class name e.g. isosceles. She tried to teach the children the properties of each class of triangle by getting them to construct representative triangles. This method required that the children understood the part played by the compass arcs and this she failed to mention. As she herself had misunderstood this as a child, it was an oversight not to explain it to her pupils. She said that her aim was to give the children practice in using the drawing and measuring instruments.

Using a Protractor

In discussion about a lesson on the use of the protractor, Jane started by blaming difficulties onto the children. Later she admitted that it was her own difficulty in understanding the protractor which was the cause of the problem.

Jane: "...I've found that some of them are really finding it hard to use the protractor. I think maybe, I'll have to go back with some of them and really start from scratch on how to use the protractor. Because although they understand that the angles [of a triangle] add up to a hundred and eighty degrees and they can work out an angle, by adding two and subtracting from a hundred and eighty, when they come to making the triangle, using the protractor, they can't do

it. You get the most wierd shapes, because they don't understand how to use it properly."

Author: "Uhum."

Jane: "With me it's usually a hit or a miss as well. But you know they are really bad some of them. So I think I will have to go back and maybe do a lesson."

Author: "What do you think it is about the protractor which confuses them?"

Jane: "I think sometimes they don't realise which end to start from and that's where I am not awfully sure myself sometimes. You know the only way that I sometimes know where to start from is that I know how big the angle should be roughly."

Author: "You mean, like it's going to be."

Jane: "Like it's going to be a hundred and ten; well obviously it's going to be much bigger than the ninety degree, you know, right angle."

Author: "Yes."

Jane: "So I know which end of the protractor to start from. But if I didn't...ah, they're starting from the other end and making it a smaller angle. But unless I think unless I know that it was going to be a much bigger angle, I would probably make the same mistake. You see, I don't really, I probably myself, I don't really know how to use the protractor properly."

Jane's strategy for angle measurement was perfectly sound, and a good way of keeping the answer sensible (using the angle's visual relation to a right angle). But, as she pointed out, it was not so

readily usable by children who were still establishing the relation between the appearance of an angle and its numerical size in degrees.

Author: "Yes well I see, em, have you thought what you might do about that?"

Jane: "No, not yet. I just discovered it this morning. Two of them were really hopeless. Because every time I gave them, just, they just did it wrong when it came to measuring. So I will have to really think of something."

Author: "Yes, em, maybe you could get hold of some protractors with only one scale on them."

Jane: "Well the ones that they're using have got the two [scales]. I think its bad for them to have the, eh, the two. Especially when they are starting off using it, because its going to get them really confused."

Author: "Yes."

Jane: "Because when I asked one of them, she said 'Oh it doesn't matter which side [of the protractor] you start from.' You know, I thought well crumbs, if she thinks that."

Author: "Yes."

Jane: "Its awfully difficult to, particularly."

Author: "Do you know why, do you know why there are two scales on a protractor?"

Jane: "No, I haven't a clue. I don't know why there are two scales on a protractor. You see I have only started using a protractor since I started teaching practice."

Author: "Did you not use one before?"

Jane: "No."

Author: "Not even when."

Jane: "I cannot remember ever using a protractor before, because I had to go to a book, you know, find out how to use a protractor."

Author: "Uhu."

Jane: "So I don't know why there are two scales on it."

Author: "Yes."

Jane: "And I don't know why, and again you see unless I know how much the angle was to be roughly, I would probably make the same mistake and start from the wrong side."

Jane revealed herself to be only a step or two in front of the class. The very issues which puzzled her are the ones which she found difficult to teach effectively. However, at least she realised what was the source of both her own and her pupils' difficulties. She had also attempted to anticipate problems, "I had to go to a book and look it up", and she tried to diagnose teaching difficulties.

Solving Equations

Jane found it difficult to solve simple equations. She brought some examples with which she had been asked to help the children by their class teacher.

Jane: "Yes, uhu, I just, I'm lost with it. I mean, I could work them out but it took me ages, but these kids seem to be doing them like that, and I, and I just can't seem to get to do it quickly."

Author: "Ok, so, so, em what c...well there are two, there are several stages for equations. I mean how do you find yourself working with these, these, these things here, er, where the thing's in words?"

Jane: "That's another thing I'm useless at as well. having to add something to twice a number. That's just, well I find that really hard."

The extract shows her worry about being slower than the children in solving equations. She mentioned that she even asked the children how they solved them but they were unable to help her.

The author asked Jane to solve a number of equations out loud. She explained satisfactorily how she would solve the following equation:

$$3n = 19 - 4$$

She then selected equations which she found difficult from the school text-book, e.g.

$$2n/3 = 12$$

She was asked to read the equation as if 'over the phone'. She replied "Well, that's two times n, divided by three is equal twelve". Evidently she understood the symbolism. But she did not understand how to start solving it. The author then posed the following similar equation:

$$m/3 = 12$$

Jane was able to solve this, but her comments showed that she believed her method to be ad hoc and lacking principle.

Jane: "Well it's thirty-six, but you'd have to say...you would say three times twelve. <u>But I don't know why you would say that</u>."

Author: "Oh."

Jane: "To get the."

Author: "All right, but, but you, yeah."

Jane: "I don't really understand why you would say, you know,

three multiplied by twelve multiplied by three."

Author: "So why did you say it: twelve multiplied by three then? I mean you are quite right."

Jane: "Ah."

Author: "Does it just seem sensible?"

Jane: "It seems sensible, but you know, I don't really know why you do it."

Here again Jane showed some knowledge of the 'rules' but was unsure of the reasons for those rules. It was difficult to know how she should be helped. She was able to read the equation, she was able to solve it, she agreed that her solution was sensible, and yet she felt unsatisfied by it. It was as if the whole enterprise lacked meaning and purpose for her. Her comment "I don't really know why you do it" implied both lack of understanding of how equations work as well as lack of understanding of what equations were for.

Extracting equations from problems stated in English proved just as difficult for Jane. She selected the following problem from the text-book:

"I am thinking of a number. If I multiply the number by 3, subtract 2 and divide by 4, the result is 4. what is the number?"

Jane: "...I just...I couldn't, I don't know where to start, Ben."

Author: "Ok."

Jane: "And another thing, even if, well, even if I did know where to start I would find it awfully hard to teach that kind of thing to a child. Well yeah, because I don't know how to teach that either, even if I did know what I was doing, and I don't."

The implication of her last statement was that even if she knew how to solve such equations she would not understand why her solution was correct. Without such understanding she would be unable to "teach that kind of thing".

4.2 INABILITY TO EXPRESS RULES

Students were often observed giving muddled explanations to their pupils. Three kinds of confusion were found. In one the student translated inappropriately from mathematical notation to English. In the second, the student had not clearly thought out what she was trying to teach. She appeared to have an imprecise mental model of the process she was trying to explain. In the third, the student mixed up two different explanatory illustrations for a mathematical concept, where each illustration had its own distinctive descriptive terms.

Translation Difficulties

Irene made an over-literal translation between the mathematical notation for subtraction and English. The following extract is taken from the start of a lesson in which she revised the subtraction algorithm.

Irene: "Now for a start we are going to do subtraction, take away. It's revision, so I don't wany anybody calling out an answer. Just put your hand up, all right. Because remember we did some tests, and I think some people were a wee bitty confused. Ok, well, we will start off with simple wee problems of subtraction. Right six from four.

Now quick."

Pupil: "Two."

Irene: "Two. Ten from eight."

Pupil: "Two."

Irene: "Two. Twenty from, from six."

Pupil: "Eh."

Irene: "See you shouldn't. Shoot you hand up."

Pupil: "Fourteen."

Irene: "Fourteen. If I had fifteen marbles and I gave Steven two, how many do I have left for myself? Gavin."

Gavin: "Thirteen."

When the tape was played back, Irene did not spot the three instances where she used "from" incorrectly. The mistake was pointed out and the extract played back again. The following conversation ensued.

Irene: "That's what I was getting at. You know when you see it written down."

Author: "Uhum."

Irene: "Em, you know like ten...even there."

Author: "Uhu."

Irene: "You know if you see it like that. I mean if you see that."

Author: "Uhu."

Irene: "You can't do it."

[she wrote down 8 - 10]

Author: "Right."

Irene: "Right, but if you see...that's what the problem, that has come up about, you know later on."

Author: "Yes."

Irene: "Uhu."

Author: "Yes."

Irene: "And I was saying it as if they were seeing it like that; do you see?"

[She wrote 10 - 8]

Author: "Oh I see what, now I see. Now I see you are translating it directly into English, sort of bit by bit: ten from eight"

Irene had made a direct comparison between the arithmetic notation and the English expression.

ten	from	eight
↕	↕	↕
10	-	8

This was pointed out as incorrect and as a source of confusion between mathematical notation and English. But as the recording proceded, we heard Irene using "from" correctly.

"...five from six leaves you with one. And one from two is one..."

But there were also further examples where she used English inconsistently and incorrectly in her descriptions of subtraction.

Irene: "Now what do we say when we see a sum like this, what do we say to ourselves?"

[the sum is 64 - 38]

Pupil: "Eight take away four."

Irene: "Eight take away four. Now can we take eight away from four?"

Pupil: "No."

Irene: "Why not? Gary, why can't we take eight away from four?"

Gary: "Because, eh, eight is four more than four."

Irene failed to pick up the incorrect pupil statement "eight take away four". She just rephrased it as "take eight away from four" and in so doing changed its meaning. This was probably a mistake since many of the pupils had committed the well known error of subtracting the upper digit from the lower when it was smaller e.g.

$$\begin{array}{r} 64 \\ -38 \\ \hline 34 \end{array}$$

Lumb (1974) observed that 1% of student teachers, whom he examined, thought that '7 - 2 = 2 - 7' even after their mathematics course in a College of Education. He offered no explanation for this misconception. The above 'translation error' may be a contributory factor.

Imprecise Model - Subtraction

A second cause of muddled explanation was the student's imprecise understanding of the process she was explaining. Irene had further difficulties with subtraction. This time they were concerned with the algorithm. She showed the children how a subtraction sum could be re-expressed to emphasise the value of each of the original digits.

$$\begin{array}{r} 364 \\ -275 \\ \hline \end{array} \qquad \text{became} \qquad \begin{array}{r} 300+60+4 \\ -200+70+5 \\ \hline \end{array}$$

Irene tried to illustrate the meaning of the technique of 'borrowing'. Properly speaking she was illustrating the method of

decomposing one 'ten' into ten 'units', rather than the equal additions method of 'borrowing' (and paying back). She again found herself in a muddle, which she openly admitted to the children. In the sum above, she showed that one could not take five units away from four units, so one would have to 'borrow'. But strangely she went on to 'borrow', not from the 'tens' but from the 'hundreds', so that the sum was transformed into the following form.

200+160+4		200+150+14
-200+ 70+5	and then into	-200+ 70+ 5

While these transformations were valid, they were somewhat confusing for the children. Irene may have anticipated the fact that the lower digit in the 'tens' column was greater than the upper digit and attempted to take a short cut. When the lesson was discussed, Irene did not have a firm grasp of the meaning of the decomposing ('borrowing') actions. The author suggested that she could have employed the bundles of ten straws which she had issued to the children, but had not used. Irene explained how such straws could have been used, taking the sum '64 - 38' as her example.

Irene: "So I would have to say, you can't...you see that's where I get muddled up. I mean, do we say well we can't take, well if I break it up from there, just dealing with my units first of all. Well I say we can't take eight away from four."

Author: "Uh."

Irene: "We've got four and we can't take eight away. Just show them, just do that. And we have to go and borrow from our, from our tens. So we take one."

Author: "Yes."

Irene: "Do we take just...one straw from the [tens] or do we take a bundle?"

Author: "What do you think?"

Irene: "A bundle of ten. No we take..."

After her last pause and evident confusion, the author fetched a box of Cuisenaire rods so that the conversation could proceed with some materials for manipulation. Irene's problem lay in recognising the value of the 'borrowed one'. She had not fully grasped that it represented ten, and would have to be a whole bundle of ten straws from the 'tens' column, and not just a single straw extracted from one of these bundles.

Irene proceeded more confidently with the rods in front of her. She layed out the following configuration (see figure 4.2) to illustrate the two numbers in the sum '64 - 38'.

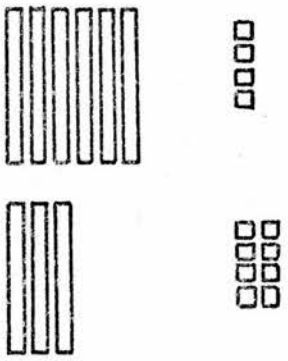


Figure 4.2. Cuisenaire Rods Modelling 64 - 38.

Irene changed the configuration to model the 'borrowing' (see figure 4.3) but did not know how to proceed. The author had to help her to exchange the 'ten rod' in the units column for ten 'unit blocks' so that the subtraction could be effected.

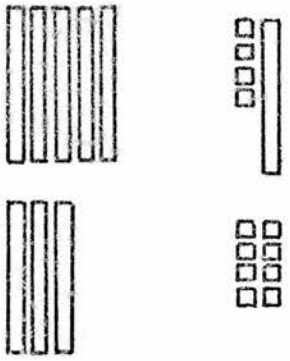


Figure 4.3. 'Borrowing' Using Cuisenaire Rods.

Imprecise Model - Fractions

The following lesson extract shows how Fiona's misunderstanding of multiplication and division of fractions produced a muddled tutorial dialogue. Fiona had drawn a disc divided into four quadrants on the blackboard (see figure 4.4)

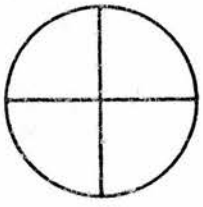


Figure 4.4. Disc Divided into Four Parts.

Fiona's intention was to get a pupil to say how a diagram containing twelve equal sectors could be produced from the diagram on the board. She focused attention on the multiplicative aspect of producing more sectors, i.e. three times as many.

Fiona: "...Now I want to make it [the diagram] into twelvths. I've got quarters. That's one quarter. what do I do to four to make it twelve? Ian, do you know?"

Ian: "Eight, you add eight."

Fiona: "You can add eight, good. What else could I do to four to make it twelve, John?"

John: "You can add, em, three fours."

Fiona: "Em, not quite, what?"

Pupil A:"I know, divide."

Pupil B:"Divide."

Fiona: "Divide four to make twelve [skeptically], Geoffrey."

This was the crux of the difficulty, and in a way the pupils were ahead of Fiona. In order to produce twelve sectors from four, each sector would have to be divided into three parts. But Fiona was concentrating on the idea that going from four to twelve was a multiplicative action.

Geoffrey:"Split it up."

Fiona: "Split it up, well yes I could. You mean split up the quarters."

Pupils: "Yes."

Fiona: "Yes, how could I do that? How many would I need to split each quarter into, Alice?"

Fiona then focused on the division aspect, that is splitting up each quarter. But as the dialogue unfolded she showed herself to be still keeping the multiplicative aspect in mind, with consequent confusion.

Alice: "Six."

Fiona: "No, what do I do to four to make it twelve? I could add eight, yes. I could not subtract, not divide, but I could do something else."

Alice: "Multiply."

[Alice has taken Fiona's heavy hint.]

Fiona: "Good. By what, John?"

Pupil C:"Three."

Fiona: "Correct."

Pupil D:"By twelve."

[Fiona folllowed up the latest pupil answer to demonstrate why it was incorrect.]

Fiona: "Right well I'll multiply by twelve. Four twelves are? Now I'll tell you; don't know your twelve-times, forty-eight. [She wrote 4x12=48 on the blackboard.] That gives me twelve, does it?"

Pupils: "No."

Fiona: "Try again; four multiplied by what gives you twelve, Tim?"

Pupil E:"Three."

Fiona: "Three, yes. Right so I am going to multiply by, my quarters by three. Eh, I've a great many, actually I've a great many more quarters don't I? So What do I do to a quarter to make it one twelvth? I don't multiply. What do I do then, Jimmy?"

Here Fiona realised that something had gone wrong. It seemed as if there were to be more quarters after the 'multiplication'. She rephrased her original question of 'what do I do to four to make it twelve' into the new question 'what do I do to a quarter to make it a twelvth'. Her next few statements had an air of relief.

Jimmy: "Divide."

Fiona: "Quite. Well done. One quarter divided by something...will give what?"

Pupil F:"Three."

Pupil G:"Three."

Fiona: "Divided by something will give you, one upon twelve. Now divided by what, Alison?"

[Fiona wrote '1/4 ÷ = 1/12' on the blackboard.]

Alison: "Four."

Fiona: "A quarter, John."

John: "Three."

Fiona: "Good, right, now we are getting somewhere. We can start dividing our quarters into equal parts."

It has taken a long time to get to this point. Fiona said later that she had a 'mental block about fractions' and was 'not too keen on them'. She said that she had taught the same lesson to a different class and that 'it had gone quite well'. Fiona's confusion stemmed from her expectation that if the diagram with four quadrants was to have more sectors then multiplication would be needed. She failed to realise that division of the quarters would produce more sectors, although each would be smaller.

Mixing Up Two Explanations

A subtraction sum, e.g. 6 - 3, can be used as an arithmetic model for at least three different situations. One is 'comparison', e.g. how much more is 6 compared to 3? Another is 'taking away', e.g. if you had 6 sweets and you ate 3, how many would be left? A third is 'complementary addition', e.g. how much must be added to 3 to make 6? Each situation is different but share a common structure which allows the same arithmetic model to be employed in each case.

A similar situation obtains for division. A sum, such as 17/3 = 5 R2, can be employed to answer two different kinds of question. One asks: if you have 17 objects and partition them into

subsets of 3 objects, how many subsets would there be? See figure 4.5.

Figure 4.5. Partition into Subsets.

Another different question asks: if you have 17 objects and share them out among 3 people, how many objects does each person receive? See figure 4.6.

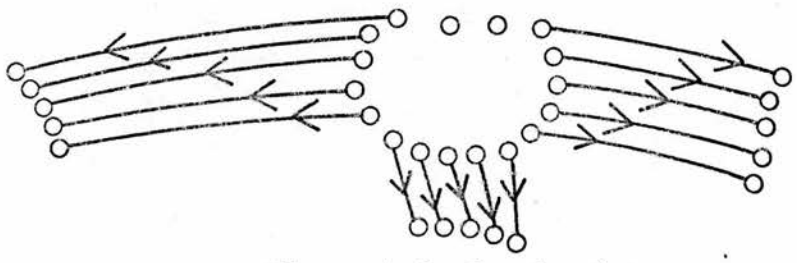


Figure 4.6. Sharing Out.

In the first case, above, the 'answer' is 5, obtained by counting the number of subsets. In the second case the answer is also 5, but this time it means 5 objects (one person's share). An explanation of division which mixes up these two different kinds of question can cause pupils to give the incorrect answer of 3. This may happen, in the first case, if the child counts objects instead of subsets, and in the second case if he counts subsets instead of objects.

Irene used words and phrases appropriate for the partitioning aspect of division while employing a concrete model of sharing. This produced a very confusing situation. She was illustrating the sum

5/3 = 1 r2. Her choice of numbers was unfortunate because they were all small and all about the same size. So there were no strong visual differences (see below) between them. In the following lesson extract, Irene was eliciting from the class how the phrase 'one and two left over' should be phrased more correctly. She introduced the word 'set' in preference to the children's suggestions of 'bundle' and 'group'. She justified this because 'set' was a shorter word. So she was expecting an answer of the sort 'one set remainder two' to her question about 'five divided by three'. This explains her rejection of the first pupil's answer (below).

Pupil A:"One remainder two."

Irene: "One remainder two. [skeptically]"

Pupil A:"Yes, three, em, remainder one, remainder two."

Irene: "Three, remainder."

Pupil B:"Two, one remainder two."

Irene: "Which is it, uh? Show me your counters."

[The pupils all had counters.]

Pupil A: [inaudible]

Irene: "Here's your counters."

Pupil A: [inaudible]

Irene: "<u>One set remainder two.</u>"

Irene's phrase "one set remainder two" referred to the <u>partitioning</u> aspect of division. But as the lesson developed, Irene started to use the <u>sharing</u> aspect. Normally this is to be welcomed because children should be aware that there are these distinct applications for division. But Irene was not aware of the distinction and was confusing the two. The children, with counters in front of them, were left unsure whether to concentrate on the

number of counters each had after sharing, or to concentrate on the number of groups of counters produced by the sharing.

Irene: "Suppose we have...I'll use the board here...now you have, here we have five counters, right, right. We want to do what...what do we want to do?"

[She wrote 5/3 on the board]

Pupil: "We want to divide them into three groups."

[Sharing]

Irene: "Yes, into three equal groups. I want to divide. We want to find out how many threes are in five.

[Partitioning]

Is that right? Right. Five sweeties and I have you, you and you, right. Now in fact we'll use...whose got some big counters? These are awful small and fiddly."

Pupil: "Me."

Irene: "Oh, yes, we'll use these ones, eh. Can I have five of these? [short inaudible phrase] Right, right, five sweeties...and I'm going to give you one, you one, right and here now what have I got left?"

[She gave one 'sweety' to each of the three volunteers - sharing.]

Pupil: "Two."

Irene: "I have got two left over. All right. Now if I give...can I do that?"

[She gave the two extra counters to two of the children.]

Pupil: "No."

Irene: "Why not?"

Pupil: "Because."

Irene: "Sh, all just try and."

[Irene had been talking to just one group of children and the rest were getting restless.]

Pupil: "Two people have got two, but one person has got one."

Irene: "That's right. So I can only give you one. I can only divide three, share three into five once with how many left over?"

[Mixture of sharing and partitioning.]

Pupil: "Two."

Irene: "Two. And we had it like this. [She rearranged the counters so each child had one.] There's one equal set, right one and one equal group, group and two left over. So that we are saying five here, it is one remainder."

Pupil: "Two."

Irene: "Two."

Irene had shifted backwards and forwards between partitioning and sharing. Her statement 'one equal set, right one and one equal group, group' was very confusing given that the counters had just been shared out. The children's difficulties were increased by her initial rejection of the answer 'one remainder two'. Irene had rejected it on the grounds that the pupil had not used the word 'set'. But because she had not explained why she did not accept the answer the pupil thought that it was because he had given the wrong numbers.

4.3 LACK OF MATHEMATICAL SELF-CONFIDENCE

The students were well aware that their understanding of mathematics was poor. This reduced their mathematical self-confidence to the extent that some of them doubted their competence in areas of mathematics which they understood.

Judging Distance

Lack of mathematical self-confidence permeated the whole of some students' attitudes to mathematics. Betty firmly believed, having been told by a 'lecturer', that she had missed some vital but unspecified mathematical experiences as a child and could now never catch up. As evidence of this, she complained that she was unable to visualise a room given its measurements, and that other people were much better at this. But as the following dialogue shows, she was quite capable of judging distance.

Betty: "...I mean even if somebody says to me, em, you know, the room was so many yards by such and such, I can't visualise that at all."

Author: "Not even approximately."

Betty: "No, the only thing I can go by is my front room. Well I take everything to my front room. But I can't really visualise again."

Author: "Do you, do you know the measurements of you front room?"

Betty: "Aha, Its about twenty-seven feet by thirty, something like that."

So Betty had an excellent method of judging distance. She compared unknown distances to the measurements of her own front room which she

knew.

Author: "Ok, so if I say 'can you try and visualise a room twelve by twelve'. What would you do?"

Betty: "Eh, well, first of all I would halve, roughly halve the length and practically third it [the breadth]."

Author: "Uhu."

betty: "But then I would have difficulty. I would have to go up to my room and sort of say well this is half. I mean I couldn't."

Author: "Can you do it now? Can you imagine a room twelve by twelve, using your room as a kind of."

Betty: [inaudible]

Author: "As a yardstick."

Betty: "Well I would have to go by what I've got in my room. Saying well that's roughly to the end of the stereo."

Author: "Ok."

Betty: "And that's roughly to the eh, middle of my settee. And em."

Author: "Yes."

Betty was able to make a judgement about a room twelve by twelve despite her protestations. The conversation continued by exploring the issue of using a personal example, her front room.

Betty: "But if it was an empty room."

Author: "Yeah, but."

Betty: "I couldn't sort of fit things in and say well, that would go in and that would go in."

Author: "Well that sounds entirely normal, I would have thought."

Betty: "Do you think so?"

Author: "Yes, I mean why, why should, I mean why should?"

Betty: "Well people trot out figures saying, you know, well, that was fifty yards and that was a hundred yards but I can't think of."

Author: "Yeah, well you can use the figures. I mean what you are talking about isn't whether you can trot them out and use them it's whether you can visualise them."

Betty: "Yes but I can only visualise it by my concrete example."

Author: "Yes, but."

This seemed to be the central difficulty. Betty was worried because she was using a completely personal image in visualising distance, not 'proper' mathematics at all.

Betty: "Now what if I never had that concrete example?"

Author: "Ok, well, what if, why do think, how do you think other people are able to do it?"

Betty: "Well I don't know. This is what I have never been able to find out."

Author: "Do you think that they don't have concrete examples?"

Betty: "Yes I think they must have a mental thing in their head."

Author: "Well, what could that be?"

Betty: "Well if somebody said to you, you know, if the room was a hundred by thirty or something. What would you immediately think of? Would you immediately think of a number, or would you think of something comparable?"

The author found it difficult to convince her that he could only visualise such a room by resorting to his own, personal concrete example.

This lack of self-confidence extended to work which the students

did in the College of Education. Both Betty and Anne mentioned that they had obtained some good marks in mathematics assignments in their course. But they felt that they did not deserve the marks because they had not really understood the material. Anne's belief in her own mathematical inability was illustrated by an incident which occurred while she and three other students were solving a problem. This was to find the number of permutations of the four letters B, A, R and G. They had all become very interested in the problem (which was set during one of the meetings in the College of Education arranged by the author). After an initial period of guessing and then enumerating the permutation possibilities, Anne found the answer. She had transformed the problem into one of finding the permutations of the four digits 1, 2, 3 and 4. She started to list the permutations in numerical order, e.g. 1234, 1243 etc. She noticed that there were six permutations starting with a '1' and six starting with a '2'. She then correctly reasoned that the total number of permutations had to be 24 because there was a "pattern emerging". The students then became interested in solving the problem for five letters or digits, and spent some time trying to find the answer. Most of the students felt that they had come across the problem before and that there was a formula for it which they had forgotten. So working from first principles they made a number of hypotheses and explored a number of different possibilities. Another student found the answer, but Anne and Betty were determined to work it out for themselves and asked her not to tell them. They tried for a little while longer and then Anne said, "This is what makes me feel like an utter failure." She saw only the outstanding unsolved problem and ignored all her sound mathematical thinking. It seemed that for

her mathematics consisted of remembering and applying the correct formula rather than the problem solving behaviour she was engaged in.

Anne wrote, in questionnaire (1):-

"I am basically afraid of maths in the class-room situation - especially at upper level. The language of maths does not come easily to me and this worries me because to teach a concept you must be precise and consistent. I have difficulty in subtracting by decomposition since I was taught to borrow 1 and pay back 1."

Anne's last sentence, about 'borrowing and paying back' is reminiscent of Irene's difficulties with subtraction. A number of other students also complained about the 'language of maths'. Irene said that she found the lessons, observed by the author, nerve-racking because she was very conscious of having to use the right mathematical words. This may explain why she felt it necessary to reject the children's use of the words 'bundle' and 'group' in favour of 'set'. She was only able to supply the children the unconvincing reason that the word 'set' was shorter. Betty also worried that when she taught mathematics, she lapsed into her own idiosyncratic language which she felt was inappropriate. Barnes (1976) has drawn the distinction between those teachers who see themselves as interpreters of knowledge and those who see themselves as transmitters. Barnes characterised the difference as followers.

"The Transmission teacher...believes knowledge to exist in the form of public disciplines which include content and criteria of performance."

whereas

"The Interpretation teachers...believe knowledge to exist in the

knower's ability to organize thought and action." (p.144)
The students behaved as if mathematics had no personal content. They were concerned to pass it on to the children in its 'proper' form unsullied by their idiosyncracies.

Jane as a Pupil

The students had not generally enjoyed or been successful in mathematics as school pupils. Jane was an extreme example. At primary school she was "...always in a complete fog..." and wrote, "No, can't remember a time when I ever liked mathematics". She remembered that she had not enjoyed "...doing mental arithmetic twice a week. + doing 'problems' eg. if 12 men take 50 days to build a wall". There was "always emphasis on being quick and accurate...which I wasn't". In secondary school things hardly improved. She gave up mathematics in the second year, started again in the third year but again gave up. She finally ended up with a Scottish 'O' grade in Arithmetic, after having "failed miserably in them all [class exams]. I remember the first mark quite clearly - 8%". Not surprisingly she could recall little that she had enjoyed in secondary school mathematics.

> "No, I hated them [mathematics topics] - far too rushed, if I could have gone at my own speed (which is v. slow in maths) I might have enjoyed it. I liked drawing geometry diagrams but having to write about them was a difficult matter. I think the great fear was in failing an exam -- great emphasis on passing and if you couldn't understand the classes, life was torture!!"

Jane's attitude remained much the same in the College of Education, "I still fear and dislike mathematics". She summed up her

difficulty as "I don't have the basic knowledge and therefore confidence". Her lack of confidence was indicated by numerous self-deprecatory remarks made during discussion of her lessons (and during the programming sessions).

> "I have got a really hopeless memory for things. Unless I keep going over things, I forget it very readily."

When she was asked what kind of things she found it hard to remember, she replied, "I think having to remember a procedure. I forget that very easily. I have to keep going on and on the whole time". Another of her difficulties, as she saw it, was that she had no 'mathematical imagination' although in other fields, e.g. art, she saw herself as imaginative. This agrees with her memory of having liked drawing geometrical diagrams; they may well have appealed to her artistically.

Lesson Preparation

Jane's lack of confidence focused on her dread of getting out of her depth in class. She dismissed her teaching, "at school (teaching practice) maths lessons were flops", or after a lesson with a new class, "I quite enjoyed it even if it was a flop".

One of Jane's strategies for coping with her 'poor memory' and with difficult mathematical topics was to prepare lessons very thoroughly. This led to its own problems. She was constantly worried lest the flow of the lesson should move her outside her plan. On several occasions it became very difficult for the author to follow the point of some of her lessons which had wandered from their original plans. For example, the children were asked a series of poorly related questions or were set activities whose purpose was

unclear.

The following extract is taken from our discussion following a lesson on fractions. Jane had been using a flannel-board to illustrate some basic ideas. (A flannel-board is a device for displaying cut out felt shapes.)

Jane: "I think from then on it kind of, I felt myself losing grip somehow. I don't know."

Author: "Em, that, your comment coincided with, with the moment you ran out of bits of felt. Up till, up till this minute you were working with your bits of felt, on the felt-board."

Jane: "I do a lot on the flannel-board. I like using it."

Author: "And that, yes but."

Jane: "And after that, I kind of lost the thread completely."

Author: "Do you think it's because, em, by actually cutting out the flannel, you think about what you are to do?"

Jane: "A lot of it yes."

Author: "Or do you think it's just, em, some other reason."

Jane: "No, I think a lot of the reason is because I like to actually do a thing, actually do a lesson, actually do a lesson before."

Author: "Yourself, you did it the night before."

Jane: "Do, I often do, you know, right from scratch."

Author: "Yes."

Jane: "Really think it out."

Author: "...do you find that in trying it out you feel that you have understood more about what you are trying to do?"

Jane: "Often yes. Because things like, like, <u>well like capacity, volume, area. I've got the vaguest notions of these,</u>

really vague...and I find that it, more in the last school I was in, I got a better idea from actually, you know, cutting out and doing different, you know, things the night before...I won't say I grasped it completely...But, you know, if the children asked me a question before I'd done the lesson, I wouldn't have had a clue what to say...But after I did the lesson, well after I had prepared the lesson, I was able to give, you know, a better...answer."

Author: "Do you think that you actually learn anything at the time you are doing the lesson? Do you ever feel, or is it that, or is that not the case?"

Jane: "No I find that I am so tense during a lesson that I am really intent on getting it across to the children...and that's as far as and, so I will say at the present moment as far as I'll go."

Author: "Yes, do, do you find that you're very intent on your lessons, em, for other things besides maths, or is it particularly maths?"

Jane: "I think for everything, but particularly maths...Particularly, I haven't got a very good memory for one thing and I find I have really got to concentrate hard to make myself remember what, what the next step's going to be...The minute I forget, the whole cor [sic, course?]."

The author established that Jane made written lesson plans. She said that she would continue to do this even after gaining her Diploma when there was no need to produce lesson plans for her tutor from the College of Education. She explained:

"Because I find that if I miss a step, that puts me in a

panic...and I like to do things step by step. I like to have the notes showing me what I am going to do next...And I think until I really get quite confident, I would like to keep that up."

The extract shows clearly how in certain topics, Jane felt herself at the limit of her resources and that only a detailed lesson plan lay between her and chaos. The reader can easily sympathise by imagining himself suddenly faced with the prospect of teaching some complex subject he knows little about. We may think of Jane as a teacher under siege.

One of the incidents which was likely to disrupt this careful planning was a well aimed question from a child. Jane explained how, on another occasion, when she did not understand angles:

"Usually I'm absolutely terrified whenever any of the children in the school asked me about angles. Because I would just have to go away and really think about it. <u>If they want an answer right there and then, I usually had to fob them off with something else.</u>"

Barnes (1973) has noted how a detailed lesson plan can act as a constraint on a teacher. He suggested that the teacher, whom he had observed rejecting a child's correct answer about fractions (see page 53) acted in this way because "this possibility seems to have been concealed from her by the admirable thoroughness of her plan". Jane was unlikely to conduct better mathematics lessons without thorough planning. She was in a difficult position which could only be remedied by a much stronger grasp of the mathematics she was attempting to teach.

Jane was not helped by some of the school teachers who

supervised her within the school selected for teaching practice. After a lesson observation, one teacher asked the author why there was concern about Jane's mathematics. The teacher maintained that Jane seemed 'keen enough'. She went on to mention that she would have loved to try all the exciting new methods in mathematics teaching but was afraid that she would get 'lost' and therefore stuck to her 'formal' methods. Another teacher thought that children should be weaned away from employing concrete materials as young as possible because she considered them babyish and tried to discourage Jane from using them. Other students found themselves at odds with the class-teachers who had a very restricted view of mathematics. Children in Mary's class were disciplined by the class-teacher for moving from their places in order to take part in a mathematical activity arranged by Mary. (Stones and Morris (1973) review the problems faced by students on teaching practice.)

4.4 A MATHEMATICS TEST

The preceding sections have painted a gloomy picture of the mathematical capabilities of the students. A mathematics test was administered which established that group 2 was not unusually poor at mathematics compared to other students. The test had been used by Rees (1974) on a sample of 108 primary school teachers. This test was chosen because both the complete test and a detailed analysis of students' responses were available (Rees, 1973; Rees, 1974). Originally the test had been used by Rees on craft and technician students. She had then extended testing to samples of student teachers and other students in higher education, so the test

contained some questions similar to those normally encountered by craft students. Most of the questions had no special 'technical' connotation, and posed such problems as the multiplication of decimal numbers and the solution of simple equations. All the questions were multiple choice and the students (in the present study) were given sufficient time to attempt all of them. This took about fifty minutes.

The mean of the scores of the students in the present study was similar to the mean of Rees' sample, see figure 4.7.

	mean score /50	standard deviation	sample size
Rees	31.7	8.5	108
Group 2	29.1	5.0	7

Figure 4.7. Comparison of Mathematics Test Scores.

The difference between the means is not great enough to reject the null hypothesis that Rees' sample and group 2 are drawn from the same population. The difference between the means is not significant at the 0.05 level using Fisher's t test (Guilford, 1957). The scores of the individual students in group 2 are given in figure 4.8.

	Score /50
Mary	36
Nina	34
Olive	31
Linda	28
Irene	27
Karen	27
Jane	21

Figure 4.8. Scores in the Mathematics Test.

Difficult Test Questions

Rees defined a 'common core' of twelve 'difficult' test questions which were answered incorrectly by more than 50% of her sample of all higher education students. This common core was expanded to fourteen questions in the case of her primary student teachers. In the present study, thirteen of these fourteen questions were also answered incorrectly by more than half the students. The remaining question was answered incorrectly by three out of the seven students. If we adopt Rees' criterion of difficulty, namely that an question is difficult if answered incorrectly by more than 50% of the students, then twenty-one of the test questions were 'difficult'. These questions are shown, in abbreviated form, in the following table (see figure 4.9). Also shown is the total number of students who answered each question incorrectly. Questions from Rees' 'common core' are marked with an '*'. The correctness of the responses of Jane, Irene and Mary to these 'difficult' questions is also shown.

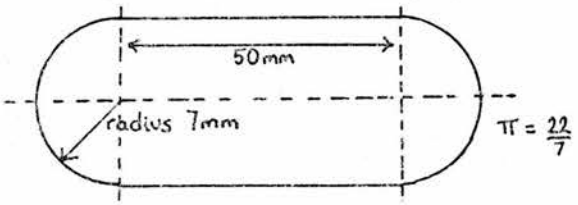
NO.	QUESTION	JANE	IRENE	MARY	TOTAL WRONG /7
		x = incorrect c = correct			
8	Calculate 0.3 x 0.3	c	c	x	4
9 *	Write 7/16 correct to 2 decimal places.	x	c	x	6
16 *	Calculate the square root of 0.9	x	x	x	7
19	If the ratio of the sides of squares is 2:3, what is the ratio of their areas?	x	c	x	4
20 *	If the ratio of the diameters of circles is 1:2, what is the ratio of their areas?	x	c	x	4
21 *	How many mm^2 in $1m^2$? (choice from 10, 100, 1000, 1000000)	c	c	x	5
22 *	How many mm^2 in $1m^2$? (choice from $10^6, 10^3, 10^2, 10^1$)	c	c	x	5
26 *	Write $5^\circ 36'$ as a decimal number of degrees.	x	x	x	7
27	Two angles of a triangle are 120° and $35^\circ 15'$. What is the third angle?	x	x	c	5
32	If 0.16X = 8, what is X?	x	x	x	5
33	How long is the hypoteneuse of a right angled triangle, if the other two sides are 9mm and 12mm?	x	x	c	4
34	Find the perimeter of the figure. 	x	x	c	4
35	Compute the height of a triangle, given its area, base length and a formula connecting them.	x	x	c	4

Figure 4.9. Difficult Test Questions.

(Continued on next page)

(Continued)

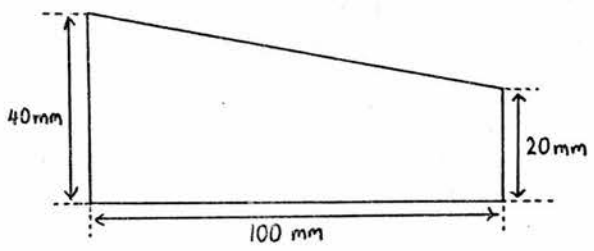
NO.	QUESTION	JANE	IRENE	MARY	TOTAL WRONG /7
		x = incorrect c = correct			
36 *	What is the area of the base of a cone given its diameter is 30mm.	x	x	x	7
37	Find the ratio of the masses of two cylinders of the same material and cross-section, if the ratio of their heights is 2:3.	x	x	c	4
38 *	Brass is an alloy such that copper:zinc is 7:3, how much brass if there is 31g of zinc?	c	x	x	5
40	Find the area of the figure. 	x	x	c	4
41 *	Find the length of metal to form a pipe of internal diameter 200mm and wall thickness 10mm.	x	x	c	6
43 *	If 1/R = 1/2 + 1/6, find R.	x	x	x	6
48 *	Find the expansion of 20mm of metal over 5°C, if coefficient of linear expansion is 0.000011 /°C.	x	x	c	5
50 *	A drill bit revolves at 600 rpm. The feed rate is 0.15mm/rev. Find the feed rate per second.	x	x	c	5

Figure 4.9. Difficult Test Questions.
(Continued from previous page)

Jane's Answers in the Test

Jane scored the lowest in the group with only 21/50 questions correct (compared to the mean for Rees' sample of 31.7). In addition to her mistakes on the 'difficult' questions, Jane answered the following questions incorrectly.

a) All the questions involving the extraction of a square root, which she failed to answer.

b) Questions involving simple internal angle properties of an isosceles triangle and a parallelogram.

c) The evaluation of '-7+8', which she answered as -1.

d) Most of the questions on area, e.g. 'a rectangle 80mm by 20mm has the same area as a square of side...?'

e) All the technical questions e.g. on heat expansion and drill feed rate.

Jane's working out, written beside the questions, showed two other weaknesses. She had to multiply powers of ten by writing the sum out on paper (as she later explained to the author, see page 87). She solved a formula rearrangement question by substituting digits for the letters, see figure 4.10.

$$8_R = \frac{V}{I} \quad \frac{16}{2}$$

$$8 = \frac{16}{2}$$

$$16 = 2x8$$

Figure 4.10. Solving for V.

Her working indicated that she grasped the underlying logic of the

manipulation but may have been uncomfortable working at the abstract algebraic level (her difficulty with equation solving was explored later, see page 93). She was able to evaluate the expression I^2R, given values for I and R. But she was unable to make V the subject of the formula

$$P = V^2/R$$

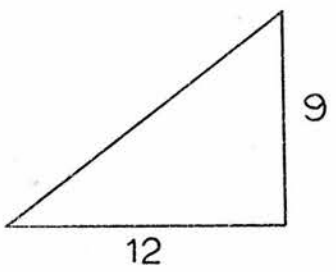
Despite her recollection of having disliked proportion questions at school, she correctly answered question 38 on proportion (see figure 4.9).

Irene's Answers in the Test

Irene gained a higher score than Jane of 27/50, though she still scored below the mean for Rees' sample. In addition to her mistakes in the 'difficult' questions, Irene made a number of other mistakes. She evaluated 8/0.16 as 2, but found X to be 1/2 in the equation 0.16X = 8 (question 32, see figure 4.9). In the problem which gave the formula:

$$\text{RESISTANCE} = \frac{\text{POTENTIAL DIFFERENCE}}{\text{CURRENT}}$$

Irene correctly calculated the resistance as 50 Ohms, given values of 8 Volts and 0.16 Amperes. All these three questions involved the same computation but she obtained three different answers. Irene's understanding of squares and square-roots was patchy. She was able to answer some of the questions involving these operations but not others. For example figure 4.11 shows her working for question 33 (see figure 4.9).



$9^2 + 12^2 = 21^2$

$\sqrt{21}$

Figure 4.11. Calculating the Length of the Hypoteneuse.

Irene, like Jane, complained that taking the test had been an unpleasant experience. She also made similar mistakes over the internal angle properties of the isosceles triangle and the parallelogram. Irene was not able to answer most of the questions on the area of plane figures correctly, but was able to find the ratio of area of similar figures, given the ratio of corresponding lengths. Her inconsistency again showed up in the way she was unable to make V the subject of the formula:

$$R = V/I$$

but was able to make V the subject of the formula:

$$P = V^2/R$$

Her working out on paper for multiplications involving powers of ten was even more long-winded than Jane's (see figure 4.12).

```
  1000
  1000
  0000
 0000
0000
1000
1000000
```

```
0000011
  x 100
0000000
0000000
0000011
000001100
```

Figure 4.12. Multiplying by Powers of Ten.

Mary's Answers in the Test

Mary obtained the highest score in the group, 36/50. She scored higher than the mean for Rees' sample. Most of Mary's mistakes were confined to the difficult questions shown in figure 4.9. The two other questions which she also answered incorrectly both involved computing square-roots.

4.5 SUMMARY

Two broad types of difficulty were found among the students. Both occurred in topics for which the students could perform written calculations correctly, e.g. simple subtraction and division of positive integers. The students' difficulties lay in explaining and illustrating the processes which they knew how to carry out. Sometimes the student had no explanation to offer at all. At other times they offered confusing explanations. There were three sources of confusion. Firstly, there were translation errors between mathematical notation and English. Secondly, students had not clearly understood the process they were explaining, but did have some understanding of it. Thirdly, students muddled up two different applications of the same process.

These and their other difficulties with mathematics lowered the students' mathematical self-confidence. Some students began to doubt their own ability even for topics which they did understand. The students' answers in a mathematics test revealed further performance difficulties. But a comparison of the scores of the students in the present study with a larger sample showed that their mathematical performance was not unusually poor.

CHAPTER 5

LEARNING TO PROGRAM

This chapter describes how the students learned to program. It examines the difficulties they encountered in planning, coding and debugging their programs. It shows that, although most of the students learned to write simple programs, they did not become competent at transforming complex problems into a form suitable for programming. The author had originally expected that students would be able to use their new programming skills to write mathematical algorithms in LOGO which would elaborate symbol manipulation rules, e.g. those for manipulating fractions. Such programs turned out to be too difficult and too time consuming for these students. Furthermore, the potential value of such programs was undermined by the discovery, described in the last chapter, that the students' main difficulty lay in understanding the meaning of the symbol manipulations rather than in defining the rules themselves.

In order to answer the question 'what difficulties did the students face in learning to program?', two kinds of analysis have been made. One is concerned with coding and the other with planning and debugging. In the first, section 5.1, the students' programs were examined and the frequency and causes of different error messages were established. This was then used to show which of the programming constructs the students found difficult to use.

A second analysis was made of the way students planned, organised and debugged their programs. This analysis is presented in section 5.2. This division of analyses partially obscures the fact that the difficulties faced by students were a complex mixture of

syntactic and semantic difficulties. The analyses also give no indication of the large amount of time that it often took students to complete their programs. To redress these issues, an annotated protocol, covering several sessions, is presented in section 5.3.

5.1 CODING

All the computer interactions of both groups were examined. The relative frequency of production of different error messages was tabulated. The analysis rests on the assumption that the error messages generated by the students give a partial indication of which programming constructs they found difficult. The context of each error was examined in order to determine its cause, the programming objective of the student at the time and the student's reaction to the error message. In judging the student's reaction, attention was paid to her typed interaction with the computer, to whether she sought help and to the length of time it took her to find the bug. This latter information was gathered from field notes made at the time and from tape-recordings of the sessions.

In addition to the relative difficulty of different programming constructs, four other factors influenced the relative frequency of production of different error messages.

(i) Different implementations of LOGO had different syntactic conventions and produced error messages under different circumstances. For example, the first two implementations of LOGO allowed a command to merely compute a value, such as:-

```
1: SUM 4 3
```

But the third implementation produced an error message under similar circumstances:-

```
W: ADD 4 3
THAT LINE WAS NOT A COMMAND BECAUSE
YOU DID NOT SAY WHAT TO DO WITH 7
```

Students made this error when moving to the third implementation. Similarly the third implementation marked non-execution of a procedure with a quote '"'. Thus the command to list a procedure became

```
W: SHOW "FRED
```

where it used to be

```
1: SHOW FRED
```

Students often forgot to include this extra quote sign when they started using the third implementation.

(ii) The teaching materials and the choice of problems placed broad constraints on the type of error that was committed. The errors reflected the ordering of programming concepts within the teaching notes. Thus few error messages were generated in the application of constructive recursion because few students progressed as far as learning this concept.

(iii) Some errors were specifically caused by the teaching strategy employed in particular notes. For instance, students were encouraged to find out, by trial and error, both the number and the data-type of arguments for a selection of the system primitives. Another problem asked students to ascertain the largest integer which the computer could store. Both these tasks produced a large number of error messages.

(iv) Sometimes error messages showed only that the student had not then been introduced to a particular programming concept. For example 'stack overflow' messages, caused by uncontrolled recursion, were usually generated by students who knew how to write recursive

procedures, but who did not yet know how to control the recursion (using a conditional and a predicate).

Error message frequencies

The programming undertaken by the students consisted of 242 sessions. In all, the students issued over 19000 commands in over 390 hours of programming and provoked over 2400 error messages, see figure 5.1. Each student response to a prompt to type was counted as a command whether she was defining a procedure or issuing a command for immediate execution. Responses to prompts issued by user-defined procedures (e.g. in quizzes) were not counted as commands.

STUDENT	COMMANDS	ERRORS	SESSIONS	TIME HOURS	ERROR RATE %	COMMAND RATE COM./HR.
Anne	1993	277	32	44.63	13.90	44.66
Betty	1868	274	27	41.95	14.67	44.53
Celia	942	126	8	10.22	13.38	92.17
Delia	114	18	2	2.67	14.67	42.70
Eve	496	57	4	6.73	11.49	73.70
Fiona	1458	168	22	36.72	11.53	39.71
Gail	2035	278	25	42.52	13.66	47.86
Harry	1098	100	17	24.33	9.10	45.13
Irene	1361	190	16	27.12	13.96	50.18
Jane	3468	332	43	83.65	9.57	41.46
Karen	358	32	3	4.75	8.94	75.37
Linda	132	33	3	4.08	25.00	32.35
Mary	3578	456	30	50.50	12.75	70.85
Nina	306	52	4	5.58	16.99	54.83
Olive	468	54	6	9.27	11.54	50.49
TOTAL	19675	2447	242	394.72		

Figure 5.1. Programming sessions.

Overall 12.4% of the commands issued resulted in an error message. This figure was calculated, crudely, by dividing the total number of error messages (for all the students) by the total number

of commands issued. The vast majority of commands issued by the students were thus syntactically correct. Cannara (1976) found median error rates of 24% and 20% in the two samples of children whom he observed learning LOGO.

It was found that 11 types of error message accounted for 96% of all the error messages received. These 11 error messages are described briefly below. Against each error message is given the relative frequency with which it occurred, rounded to the nearest percent. The relative frequency was calculated by dividing the total number of errors of a particular type by the total number of all errors. The relative frequencies of the errors are summarised in figure 5.2.

1. Call undefined procedure	28%	**********
2. Insufficient arguments	16%	**********
3. No line number	11%	**********
4. Extra text	10%	**********
5. Turtle off drawing area	10%	**********
6. Variables misused	6%	******
7. Wrong type of argument	4%	****
8. Command leaves a value	3%	***
9. Device claiming violation	3%	***
10. Number too large	3%	***
11. Stack overflow	2%	**

Figure 5.2. Most Frequent Errors.

The error messages are described, most frequent first. The wording in the headings is not the wording of the actual error messages.

1. Attempt to use an undefined procedure(28%)

   In LOGO unpunctuated character groups were interpreted as procedures to be executed. The most frequent error detected by

the system was its attempt to execute a procedure which was not defined. Errors due to the attempt to list or save undefined procedures are also included in this category.

2. Insufficient arguments(16%)

Many programming errors produced the symptom that a procedure was not given sufficient argument values.

3. Line number forgotten(11%)

Surprisingly, this error was the third most frequent. It was produced when the student forgot to include a line number in front of a command when she was editing or defining a procedure.

4. Extra text in a command(10%)

This error was produced when the LOGO interpreter parsed a command but found more text in the command than it needed (in its attempt to match arguments to procedures).

5. Turtle off edge of drawing area(10%)

Much of the students' drawing was output to either the storage-tube display or to the graph-plotters. If an attempt was made to move their simulated turtles outside the drawing area an error message was generated.

6. Variables misused(6%)

This category included a number of different error messages, related to misuse of variables, such as the attempt to access an undefined variable, the misuse of a procedure as if it was a variable and misuse of a literal as if it was a variable.

7. Wrong type of argument(4%)

This error message was generated when a procedure was given an argument of the wrong data-type.

8. Command leaves a value(3%)

In the third implementation of LOGO it was illegal to issue a command which merely computed a value.

9. Device claiming violation(3%)

This error message was generated if the student attempted to draw without first having claimed a drawing device, or tried to claim a drawing device already in use.

10. Number too large(3%)

This error message was generated when the student attempted to input or compute an integer greater than the computer could hold (i.e. >8,388,607).

11. Stack overflow(2%)

This error message was generated when the number of nested procedure calls was greater than the stack depth, e.g. in uncontrolled recursion.

The error message frequencies for individual students were broadly similar. The main exceptions were those students who attended less than about 10 sessions. For these students (see figure 5.1) the programming content of their early sessions distorted their error message frequencies. For example Nina and Linda spent much of one of their few sessions finding out the largest integer they could input to the computer. These two students produced 40 (out of their total of 84) error messages reporting 'Number too large'. This large number put this error message in tenth place in the frequency chart (see figure 5.2). The error message frequencies for the 8 students who attended 10 or more sessions are given as figure 5.3. Only Harry's chart shows marked departure from the norm. He also committed fewer errors than average.

In general, individual student error rates were remarkably

similar. The mean error rate for the eight students who attended more than 10 sessions was 12.4% (calculated by taking the average of the individual student rates). The standard deviation was 1.97%. Harry and Jane committed fewer errors than the average whereas Anne, Betty and Jane committed more. But there is not a great difference in error rate between Harry who had the lowest rate (9.10%) and Betty who had the highest rate (14.67%).

Rates of working were also broadly similar with a mean of 48 commands per hour for the 8 students who attended more than 10 sessions. This was calculated by averaging the individual rates of working of the 8 students. Mary was the major exception with a much faster rate of 71 commands per hour (see figure 5.1). The students who attended less than 10 sessions were not included for the same reason as before, namely that the content of one or two of their few sessions distorted their rate of working (generally making it look faster).

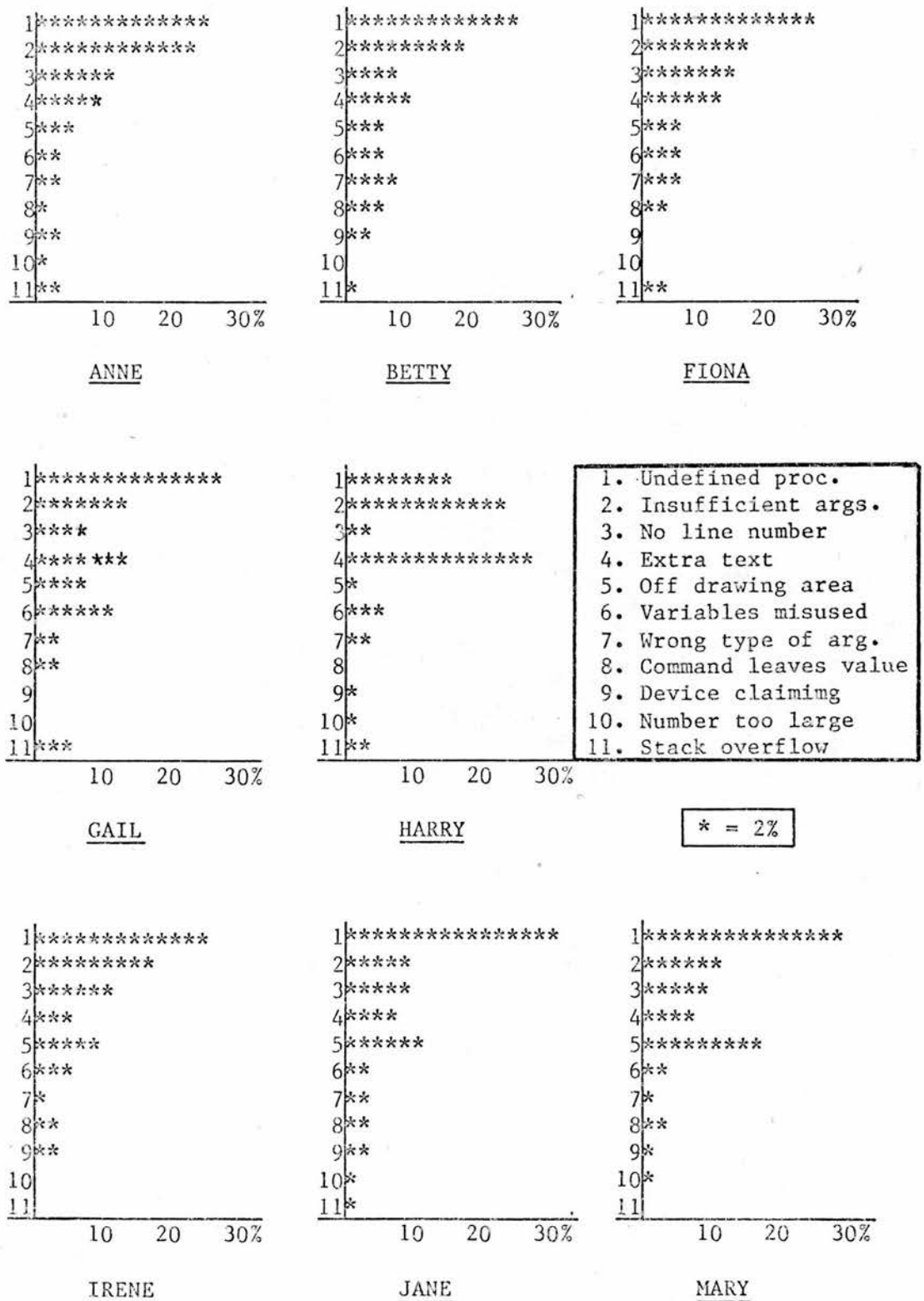


Figure 5.3. Individual Error Message Frequencies.

Causes of Errors

Cannara (1976) characterised the major difficulties of children learning LOGO as "ill-defined intents" and as a "misunderstanding of linguistic/computational context". Both these phrases can be applied to the adults studied in this thesis. This supports Statz's (1973) contention that adults make much the same mistakes as children when learning LOGO. Though the students in the present study produced fewer errors than the children in Cannara's experiment.

Many of the errors committed by the students betrayed a temporary lapse of concentration or a failure to take all the syntactic rules into account. The students exhibited "ill-defined intents", but these did not generally arise from deep-seated misunderstandings of the language. Thus although students often needed help to find the cause of an error message, they could usually repair the mistake once it had been pointed out. Part of their difficulty in finding causes for errors was the mismatch between the error message and the cause of the error. The error message referred to the syntactic violation from the point of view of the virtual machine rather than from the programmer's point of view. Certain error messages were particularly unhelpful, especially in implementations 1 and 2. For example, the following error message was generated by a student using argument values in the format of argument names, when trying to run a drawing procedure, which she had named FORK and which contained no arithmetic commands:-

```
1: FORK :100 :45
[UNDEF * 0.0871] ONLY NUMBERS CAN BE USED IN ARITHMETIC
```

The student should have typed :-

```
1: FORK 100 45
```

Later we give further examples of totally misleading error messages which 'assisted' the student to make further, worse, errors.

The experience of these students emphasises the importance of clear and understandable error messages. These should refer to the machine in the same terms as have been introduced in the teaching materials. Considerable efforts were made in the third implementation of LOGO to fulfil these criteria.

Misunderstandings of the computational context were frequently observed among the students. But they often seemed to be caused by the student forgetting to inform the machine that the context was to be changed. This seemed more an oversight than a misunderstanding and seemed to be caused by the student treating the machine either as intelligent and human or as an extension of herself. In the latter case, the student knew that she wanted to edit, say, instead of issue a command for execution and took it for granted that the machine was aware of her 'unvoiced' intention to change context. After a computer failure students would forget that their workspace had been erased and try to continue as if nothing had happened e.g. by attempting to draw with the same device, or by referring to some previously defined procedure. These errors seemed to indicate a temporary failure to distinguish between their own and the machine's view of the session.

Students had little difficulty with the filing system, as their lack of errors indicate. Apart from the difficulties mentioned already, most students found the simple line editor easy to use. This contrasts with Austin's (1976) observation about the adults he taught and is probably accounted for by the greater simplicity of the filing and editing system employed in the implementations of LOGO

used in Edinburgh.

In the rest of this section the causes of the eleven most frequent error messages are described. The context of each error was examined to see what had caused the error message and to establish whether it was due to, say, misunderstanding or mistyping. This distinction required knowledge of what the student was attempting, what the student subsequently typed and whether the student asked for help. In some cases it was impossible to establish the cause of an error, for example, if the student immediately abandoned the problem and started on something else. This might give grounds for suspicion that the student was completely stuck, though it might also indicate that she was bored with the problem and that the error provided a convenient moment to stop.

For each error message, the causes are discussed, most frequent first. The frequency, given in brackets after each cause, is the relative frequency of that cause rounded to the nearest percent. It is calculated by finding the total number of error messages of the given type (from all the students) and then comparing different causes within this total.

1. Attempt to use an undefined procedure

Many different reasons for this error were found, only one of which could be attributed to the student's deliberate attempt to run a procedure which was not in her workspace.

(i) Mistyping not involving a space character (26%)

The most common cause for the error was simply mistyping e.g. 'FOFWARD' instead of 'FORWARD'.

(ii) Mistyping involving a space character (22%)

Misuse of the space character caused a large number of errors. In LOGO a space character acts as a delimiter. In this respect several spaces were treated as if they were a single space, but inserting a space or forgetting a space often caused an error message. For example, students would type 'PEN UP' for 'PENUP' or conversely 'FORWARD100' for 'FORWARD 100'. In most cases students were able to establish the cause of the error without help and fix it themselves. The first two implementations of LOGO seem to have been particularly conducive to this error because of the number of concatenated two word primitive names e.g. 'PENUP'. To make matters worse there were also two word primitive names which were not concatenated e.g. 'LOAD SAVED'. The second part of this name, 'SAVED', was not the independent procedure call which LOGO syntax would normally lead one to expect. So Fiona's sensible attempt to list the names of her saved procedures failed when she typed:-

```
1: PRINT SAVED
```

The third implementation of LOGO removed such double-barrelled names in favour of single word names.

(iii) Procedure not in workspace (19%)

Students did attempt to run, list or change procedures which they had forgotten to define or to load into their workspaces. This would happen, especially, after a crash when the student would forget that the crash had destroyed the contents of her workspace. This was a case where the student temporarily forgot that the virtual machine's context was different from her own.

(iv) Missing punctuation e.g. '"',':' or '[]' (15%)

Another cause of this error message was the student forgetting to quote a literal or a name. Many students had difficulty

distinguishing between referring to a procedure by name (with a quote) and asking for the procedure to be run (no quote). This difficulty was increased by the change to implementation 3.

(v) Using a synonym (5%)

Students occasionally typed a synonym instead of the required procedure name e.g. 'TURN' instead of 'RIGHT' or 'LEFT' and 'BACK' instead of 'BACKWARD'. Occasionally they would forget the name of a procedure, which they had defined, and call it by a new name e.g. 'OCTOGON' instead of 'HEXAGON' or 'COUNTDOWNANDUP' instead of 'COUNTUPANDDOWN'. Mary used 'THEN' as an infix connective for two commands in the sense 'and next' rather than as part of the conditional 'IF...THEN'. She typed a line consisting of '<command> THEN <command>', having used an inappropriate English connotation of 'THEN'.

(vi) Wrong context (4%)

Occasionally students misread, or failed to read the prompt, and used a procedure in the wrong context, for example, by attempting to delete a procedure line while still at command level. Some students typed 'END' at command level (it should be used to exit from procedure definition mode). They appeared to be trying to use it as a kind of interrupt. The correct method of interrupting execution in the third implementation was baroque. It consisted of pressing escape, typing 'Q' in response to the resultant prompt 'INT:' and then pressing carriage return twice. Some students tried to interrupt execution by typing 'Q' at command level rather than at interrupt level.

(vii) Other causes

Students mistook parentheses () for square brackets [], and vice

versa so that a list of literals became an expression containing procedure calls. Some students were confused by the meaning of the quote sign '"' and used it in front of procedure calls which should have remained unquoted. Some students deliberately typed in facetious English comments which they knew would generate error messages. This relieved the frustration of problem-solving. Occasionally students attempted to retrieve procedures from their long term file storage which they had not saved.

2. Insufficient Arguments

There were four main causes of this error message. Some of causes had little to do with the student misunderstanding the number of arguments a procedure needed.

(i) Incorrect command level call (40%)

Students often forgot to give argument values when they called a procedure at command level. This could happen even in a long sequence of otherwise correct calls to the same procedure and seemed often to be merely a slip. Students sometimes forgot to give an argument value just after editing or defining a procedure which expected arguments. This looked as if their attention was wholly given over to the desire to run the procedure as soon as possible, forgetting that argument values would be needed as well.

(ii) Finding out about the system (27%)

Explicit instruction in one of the worksheets asked the students to use trial and error to find out how many arguments certain system primitives needed. This process produced a large number of errors.

(iii) Incorrect sub-procedure call (19%)

The smaller frequency of errors in sub-procedure calls (within a

procedure definition) compared to command level calls may be due to greater care being taken during procedure definition than at command level. During definition, the lines of the procedure were not executed and so there was not the same impatient anticipation to see the effect of the command as was suggested as cause (i). Typical errors were to forget that a recursive sub-procedure call required the same number of arguments as the super-procedure or to forget to supply argument values to a procedure iterated with a 'REPEAT' For example:-

REPEAT 4 FORK

instead of

REPEAT 4 FORK 100 45

(iv) Failure to quote procedure names (9%)

Another cause of the error message, in implementation 3, was for the student to forget to quote the procedure name to which she wished to refer. Students tried to list a procedure by typing

SHOW PEAK

instead of

SHOW "PEAK

where procedure PEAK expected arguments. This error was probably produced because of the change over from one implementation to the other.

3. Line number forgotten

Two causes were found for this error. This was one of the few errors in which the error message correctly described the cause of the error as distinct from the symptom, by typing 'did you forget the line number'.

(i) Just forgot or mistyped line (75%)

In the majority of cases, the student seemed merely to have forgotten the line number or to have mistyped the line so that the line number was not picked up by the machine e.g.:-

either

```
D: FORWARD 100
```

or omitting a space character

```
D: 10FORWARD 100
```

instead of

```
D: 10 FORWARD 100
```

(ii) Wrong context (24%)

In other cases the students sometimes attempted to give a command while in edit mode, having forgotten or misjudged the context. Sometimes they would try to repair an earlier editing mistake by reentering the editor while already in edit mode. A typical example was when the student realised that she had forgotten to include an argument name in the title line of her procedure. She realised that an edit was needed but forgot that she was already in the appropriate mode to carry out that edit and instead tried to reenter edit mode.

```
1: EDIT WHEEL
&: EDIT WHEEL :SIZE
YOU FORGOT THE LINE NUMBER
```

In this implementation the student could have included the argument name by typing the following sequence of commands:-

```
1: EDIT WHEEL
&: TO WHEEL :SIZE
&:
```

Confusingly, use of 'TO' in edit mode was interpreted as redefining the title of the procedure. At command level use of 'TO' with a

procedure name, which already existed, was interpreted as a complete redefinition of the procedure (thereby throwing away the old definition). Students often deleted their definitions inadvertently by using 'TO' rather than 'EDIT' in this way. It may have been to avoid any possibility of overwriting an existing definition that students used 'EDIT' rather than 'TO' in edit mode. This difficulty was reduced in the third implementation of LOGO which provided a special primitive ('RETITLE') for changing procedure title lines and did not allow students to inadvertently delete their definitions in this manner. It gave an error if the student attempted to define a new procedure using the name of an existing procedure.

4. Extra text in a command

Four major causes of this error were in evidence.

(i) Finding out about the system (45%)

The main cause of this error message was the worksheet which asked students to use trial and error to determine the appropriate arguments for certain primitives.

(ii) Poorly formed commands (33%)

A common cause of this error was the construction of poorly formed commands. Sometimes students attempted to do two actions in a single command e.g.:-

```
&: 10 PRINT FIRST :LISTA FIRST :LISTB
```

Sometimes they supplied more argument values than were needed, e.g.:-

```
&: 10 LEFT 50 360
&: 10 TEST EQUALQ 0 :NUMBER 0
```

Fixing these errors often required the help of the tutor. Genuine difficulties with the LOGO syntax rather than just slips or

mistypings were often indicated. This difficulty could probably be reduced by explicitly teaching the students to delimit expressions with parentheses. The mistaken command (above) would become

&: TEST (EQUALQ 0 :NUM) 0

and the extra '0' would be more obvious. Without parentheses, LOGO's prefix commands are often very difficult for the human to read. Further examples are given in section 5.3.

(iii) Procedure argument names included, in wrong context (12%)

Students sometimes forgot or misunderstood that they should not include a procedure's argument names when commanding it to be listed or copied, i.e.

W: SHOW "SQUARE "SIZE

where 'SIZE' is the parameter of procedure 'SQUARE'. This difficulty was augmented by the teaching which identified the name of a procedure and the names of its formal parameters as a single unit, the 'title line'.

(iv) Line number given at command level, wrong context (5%)

Students sometimes forgot wich mode the machine was in and typed in commands with line numbers when the machine was at command level as if they were editing or defining.

5. Simulated turtle off drawing area

Two causes were found for this error message.

(i) Misjudged turtle movement (66%)

Students misjudged how large their drawings were going to be or started the simulated turtle off too near the edge of the drawing area. Sometimes students drew in an 'exploratory' fashion (see section 5.2) e.g. by repeating the execution of a procedure to see

what it would draw. This process would sometimes terminate with this error message.

(ii) Uncontrolled recursion (33%)

Students would often let recursive drawing procedures run until the simulated turtle attempted to move outside its drawing area, unless a stack overflow error message occurred first.

6. Variables misused

Although errors of variable misuse were not the most frequent, they did cause the students much difficulty to debug. This was because the mechanisms of assignment and argument binding were hidden and because error messages were either unhelpful, or in some cases, positively misleading. Five causes for this error message were found.

(i) Title line in wrong format (40%)

Many students found the idea of defining a procedure which could take an argument hard to grasp. They did not realise that when defining the procedure, the argument could only be referred to by name, but when running the procedure the argument value should be supplied. Students would sometimes supply a typical value instead of a name at define time and they would confuse the punctuation rules for names and values (literals) both in the title of a procedure and in its body.

Students sometimes set out procedure title lines in the wrong format. Sometimes this as due to a mistyping, e.g.:-

```
1: TO HEXAGON ::SIZE
```

At other times, they muddled up argument names and argument values and attempted to supply values instead of names at define time e.g.:-

```
1: TO GROWCIRC :SIZE 360
```

or

```
1: TO EQUIVER :NUMBER [X Y]
```

In the latter example, the form of the second argument, '[X Y]' reflected the form of a typical argument value '[2 5]', where 'X' and 'Y' are used as variables. Further errors were caused in implementation 3 where students forgot to quote the procedure name or left a space between the name and the quote sign.

(ii) Variable called but not declared (24%)

Students would define a procedure without an argument but refer to an argument in the body of the procedure. This was partially caused by their habit of planning procedures at the teletype. Some students tried to use global variables which had not been assigned values.

(iii) Argument values in the format of names (14%)

In the first two implementations of LOGO, argument names in a procedure title were preceded by a ':' character. The same symbol was used in the procedure body to retrieve the value of the argument, e.g.:-

```
1: TO SQUAREBIT :SIZE
&: 10 FORWARD :SIZE
&: 20 LEFT 90
&: END
```

Students sometimes used the symbol ':' incorrectly, in front of the argument value when running a procedure e.g.:-

```
1: SQUAREBIT :30
```

or confused ':' with '"' as follows:-

```
1: CHOP :FORMATION
```

instead of

```
1: CHOP "FORMATION
```

A common cause of this mistake was a particularly misleading error message. This message resulted when the student had defined a procedure in which the title line was incorrect because the proper format for arguments had not been used. A typical example of this 'computer assisted error' is as follows. Gail wanted a procedure which would type a given letter a given number of times. There are a number of mistakes in her definition, but the important one, for present purposes, is the incorrect format of the title line (using "WORD instead of :WORD).

```
1: TO TRYOUT :NUMBER "WORD
&: 10 TYPE :NUMBER SPACE
&: END
```

Gillian then attempted to get 6 W's typed.

```
1: TRYOUT 6 "W
INPUTS MUST BE PRECEDED BY :
CHECK YOUR PROCEDURE
1: TRYOUT :6 "W
INPUTS MUST BE PRECEDED BY :
CHECK YOUR PROCEDURE
1: TRYOUT :6 :W
INPUTS MUST BE PRECEDED BY :
CHECK YOUR PROCEDURE
```

Her first attempt to call TRYOUT had been correct and subsequent incorrect calls were her reasonable response to the misleading error message. In the third implementation, the format of procedure titles was checked at the time of definition instead of execution.

(iv) Mismatch between name in the title and in the body (9%)

Sometimes students used one name for the argument in the title of their procedures and a different name in the body of the procedure, e.g. 'NUMBER' and 'NUMBERS'.

(v) Procedure used as if it was a variable (4%)

The third implementation of LOGO had two main kinds of

user-defined and system procedures. One acted as a function and returned a value ('gave a result'). The other was executed for the sequence of side-effects it produced and did not return a value. There was also a hybrid form which did both. The first two implementations of LOGO, following LISP, did not distinguish these forms clearly since every procedure returned some value. Consequently many students found the distinction hard to understand and difficult to apply. Occasionally students used a procedure, which returned a value, as if it was a variable. While this was an infrequent occurrence, students found the error hard to understand. For example, one student, having just transferred from implementation 2 to 3, typed:-

W: RECALL :REMEMBERED

instead of

W: RECALL REMEMBERED

where 'REMEMBERED' is a procedure which returns a list which can be the argument value for 'RECALL'. She had typed ':REMEMBERED' as if it was a variable.

Another student was provided with a procedure called 'CHOOSE' which returned a four-digit random number. She attempted to assign such a random number to a variable named 'BOX' by typing:-

W: MAKE "BOX VALUE "CHOOSE

instead of

W: MAKE "BOX CHOOSE

She then tried to see what value 'CHOOSE' would return by typing

W: PRINT VALUE CHOOSE

instead of

W: PRINT CHOOSE

Students did not see the difference between defining a procedure which was to 'PRINT' a value and defining a procedure which was to return a value. This difficulty was augmented because the primitive (in the first two implementations of LOGO) used to return a value was named 'OUTPUT'. The ordinary connotation of its name was similar to that of 'PRINT' and appeared to refer to printed output.

In the third implementation the name 'OUTPUT' was changed to 'RESULT' to try to emphasise the internal flow of information. A clear distinction was drawn between procedures which returned a value and those which did not, both in the implementation and in the teaching materials. Procedures which formerly had returned values, such as 'PRINT' and 'MAKE' no longer did so. User-defined procedures only returned a value if specifically requested by the programmer, and not by default.

(vi) Other difficulties with variables

Another difficulty with variables concerned the hidden mechanism of argument binding which makes use of positional information. Students often failed to understand the local scope of argument names. For example, they would use the same name for the arguments of every procedure in a hierarchy of super and sub-procedures as if the argument values were communicated from the super to the sub-procedure through their sharing a common argument name. Difficulty with argument binding was most evident in students' attempts to understand the elegant, but highly compact, mechanism of recursion. Consider the following recursive procedure:-

```
DEFINE "SPIRAL "SIDE "ANGLE "STEP
10 FORWARD VALUE "SIDE
20 LEFT VALUE "ANGLE
30 SPIRAL (ADD VALUE "SIDE VALUE "STEP) (VALUE "ANGLE) (VALUE
 "STEP)
END
```

The notion that each recursive call was quite distinct and created a new local context (e.g. for argument values) caused difficulty because the names of the recursive sub-procedure and of its arguments stayed the same as the super-procedure. Students muddled the context of the super-procedure with that of the sub-procedure. In reading line 30 of 'SPIRAL', from left to right, the student first of all read the recursive call to 'SPIRAL', and secondly read the expressions which evaluated to its argument values. This gave the impression that these expressions were evaluated in the context of the recursive sub-procedure call instead of in the context of the super-procedure.

To try to counteract this misunderstanding, students were given multiple copies of the recursive procedure in question (by listing it at the console). The copies were separated and used to hand trace through a nested sequence of calls (to 'SPIRAL', say). In each call of 'SPIRAL', all instances of argument names were overwritten with their values by the student with a pen. This was done before the student traced through that particular call of the procedure. Thus the recursive call in line 30 would already have the correct values for the argument names written in.

7. Wrong type of argument

Four main causes for this error message were found.

(i) Finding out about the system (40%)

The most common cause of this error message was the worksheet which asked the students to use trial and error methods to determine the arguments of certain primitives.

(ii) Forget 'VALUE' (15%)

Students used the name of a variable instead of its value by omitting to type 'VALUE'. That is, they typed merely '"X' instead of 'VALUE "X'. Difficulties with the name/value distinction have already been discussed.

(iii) Difficulties with 'REPEAT' (11%)

Students made a variety of mistakes with the iterative control 'REPEAT'. It took two arguments, a positive integer and a command, and repeated the execution of the given command that number of times. Students sometimes omitted the first argument, the number. The order of the arguments was counter-intuitive, e.g.

REPEAT <number> <command>

and some students inverted their order. Students also forgot to give the command to be repeated in full, typically leaving out an argument value.

(iv) Uncontrolled recursion (9%)

Occasionally recursive procedures were written which were not controlled or controlled incorrectly. If the procedure was recursively searching a list, say, this could lead to the attempt to extract the first element of an empty list, which would cause an error.

(v) Other causes

Another cause of this error was confusing parentheses and square brackets. Students sometimes defined procedures which requested input from the console using the primitive 'REPLY'. Typing the wrong kind of input would cause this error message.

8. Command leaves a value

Three main causes of this error were found.

(i) Forgetting to PRINT the value (29%)

A frequent cause of this error message was the student forgetting to print the value returned by a function at command level, e.g.:-

W: ADD 4 3

instead of

W: PRINT ADD 4 3

(ii) Inserting a '"' in a procedure call (28%)

Another cause of this error was quoting a procedure call. In implementation 3, the quote sign in front of a procedure name indicated that the name of the procedure was intended and inhibited execution of the procedure. Students misunderstood this and inserted quote signs in front of procedure names which were to be executed.

(iii) Wrong context (20%)

Another cause of this error was the student typing a command with a line number at command level as if she was defining a procedure. This error has already been commented on under error message number 4.

(iv) Other causes

Some students, who were calling the same procedure over and over at command level, forgot to type its name and only typed the new values they had selected for its arguments. It was as if they expected the system to realise that they were referring to the same procedure as before and did not need to explicitly mention the point.

9. Device claiming violation

Three main causes for this error were found.

(i) Device unallocated because of crash (59%)

After a computer failure, students sometimes forgot that they needed to reclaim a drawing device if they wished to continue drawing.

(ii) Claim device already in use (20%)

Another cause of the error was the students' attempts to claim a device which had already been allocated.

(iii) Forget to claim device (20%)

Students sometimes forgot to claim a drawing device, having spent the first part of a session defining new drawing procedures. This seemed a simple oversight on their part.

10. Number too large

The predominant cause of this error message was the problem given to the students of finding out the largest integer which the computer could store.

11. Stack Overflow

Two causes for this error predominated.

(i) Uncontrolled recursion (83%)

The most frequent cause of this error was the deliberate execution of a recursive drawing procedure which contained no stop rule. Students were introduced to recursion before conditionals and predicates and so were able to define recursive procedures before they knew how to control the recursion.

(ii) Inadvertent recursion (17%)

A second cause of the error was the student's accidental definition of a recursive procedure. This sometimes happened when the student forgot, or misunderstood, which procedure she wished to call as a subprocedure.

Summary of Coding Difficulties

The error messages produced by the students were examined. Eleven error messages accounted for 96% of all errors. The error rates for different students were similar. The students' most persistent difficulties were concerned with variables. There were also difficulties with syntactic markers, especially the quote sign. Mistyping caused many errors, especially mistyping involving the space character. Unparenthesised expressions were also a source of difficulty. Broad similarities were found between the errors of the students and children's errors described by Cannara (1976), e.g. errors of computational context. This confirmed Statz's (1973) conjecture that adults make the same kind of mistakes as children when learning LOGO.

5.2 PLANNING AND DEBUGGING

This section describes some of the planning and debugging strategies used by the students when constructing programs to draw pictures. Such programs were used extensively in the programming syllabus. Drawing with the turtle provided a convenient method of introducing many programming concepts. But it encouraged students to adopt problem-solving strategies which did not transfer readily to other classes of problem. Five strategies were differentiated: 'direct driving', 'linear refinement', 'incremental', 'exploratory' and 'limited top-down'. The first four of these strategies are all variants on a bottom-up approach. Students produced programs by concentrating on low-level details rather than by working to a systematic plan. It is necessary to distinguish program organisation from the method used to construct the program. It is possible to

write a well-structured program (i.e. where the functional relation of the parts is reflected in the program organisation) in a bottom-up fashion, if minor sub-procedures happen to be defined before the main procedures. However, the programs written by the students were not usually well-structured, were not planned systematically and were often written by defining sub-procedures before super-procedures.

The five strategies (above) were not uniquely ordered as stages through which each student passed. Though 'direct driving' was usually adopted by novices and 'limited top-down' by more experienced programmers. These strategies enabled the students to solve drawing problems without conducting a detailed preliminary analysis of the problem. This meant that they were not well equipped to deal with other classes of problem e.g. the fraction manipulating procedures described in section 5.3. The strategies also betrayed the students' preoccupation with the drawings, which their programs produced, in contrast to the process of producing the drawing. On the whole the students were more concerned to see the (drawing) products of their work than to analyse the structure of the drawing or improve the efficiency or elegance of their programs. This view of problem-solving clashed with that which the author had expected to teach and caused a number of difficulties later in the mathematical work which the students undertook. For example, they often adopted a trial and error approach rather than an analytical approach to problems.

Direct driving

The title, 'direct driving' refers to the way that many students planned their drawings at the terminal, not by defining procedures, but by issuing commands for immediate execution by the turtle (or the

simulated turtles). This strategy seemed to be due to two factors. Students were often more interested in the drawings than in the process of programming. Students took some time to learn that they could considerably extend their programming powers by defining procedures. At first, defining procedures appeared counter-productive because of the difficulty students had in working out the effect of each command on the turtle. This is because in definition mode, commands are not executed and the programmer has to anticipate what their effects will be. This process was made more difficult because the students liked to work at the console and preferred not to write down a plan on paper first.

The students had to learn that a sequence of commands, written as a procedure, could be debugged more quickly because it eliminated the need for tedious retyping. They also had to learn that a procedure, once defined, could be used as a unit in other problems. Finally they had to learn that a procedure could be used as a new 'primitive' which enabled much more complex programs to be built. None of these three properties of procedures was self-evident and each took time to be appreciated by the students.

A common occurrence was for a student to produce a drawing by issuing drawing commands and by reworking the sequence of commands (at command level) until the drawing was satisfactory. Only when the drawing was correct would the student define the debugged sequence of commands as a procedure. This was often done just to preserve that particular sequence e.g. in order to show the drawing to others. Sometimes in the course of debugging such a sequence, the student would turn the turtle in the wrong direction, say 'RIGHT 60' instead of 'LEFT 30'. She would counteract this by issuing a further

command, 'LEFT 90', which would both undo the effect of the incorrect command and turn the turtle the correct amount. This pair of commands e.g. 'RIGHT 60, LEFT 90' would sometimes be included, unchanged, when the student reissued the whole sequence as she debugged the picture. They would be included unchanged, instead of as the single command 'LEFT 30', even in the procedure which she would finally define.

Students would sometimes include their own procedures in their direct driving if they were very familiar with their effects. The sequence would be debugged as before, by reissuing it with corrections, but would contain calls to both primitives and user-defined procedures. Again only when the student was satisfied with the result would she define the whole sequence as a super-procedure.

Super-procedures produced by this method were not deeply hierarchic, typically calling only one level of user-defined procedure.

Linear refinement

Linear refinement was another bottom-up strategy used by the students. In this strategy the student would define a procedure which included only those commands which she was fairly sure were correct. The procedure was then tried out and corrected if necessary. Then the procedure was expanded by adding a few more commands and then tested again in the same way. Using this method the procedure gradually grew until it solved the whole problem.

A variation on this method was to define a new procedure at each stage whose content was identical to the last procedure defined, except for the addition of the new and untested commands. Procedures

defined earlier in the debugging sequence were then abandoned. Irene was observed using this extremely inefficient method. She was trying to draw a diamond shape. The protocol (below) has omitted six lines in which Irene either forgot a line number or which she overwrote later in the same definition.

```
W: DEFINE "GIRL
   D: 10 LEFT 40
   D: 20 FORWARD 60
   D: END
GIRL DEFINED
W: GIRL
```

Irene cautiously defined and ran the first part of the program.

```
W: DEFINE "BOY
   D: 10 LEFT 40
   D: 20 FORWARD 60
   D: 30 LEFT 40
   D: 40 FORWARD 60
   D: END
BOY DEFINED
W: BOY
```

Procedure 'BOY' extended the program but copied procedure 'GIRL'.

```
W: DEFINE "SHAPE
   D: 10 LEFT 40
   D: 20 FORWARD 60
   D: 30 LEFT 40
   D: 40 FORWARD 60
   D: 50 LEFT 40
   D: 60 FORWARD 60
   D: END
SHAPE DEFINED
W: SHAPE
W: CENTRE
```

Procedure 'SHAPE' was not quite right so Irene put the turtle back in the centre of the screen ready to debug the program.

```
W: DEFINE "HAND
   D: 10 LEFT 40
   D: 20 FORWARD 60
   D: 30 LEFT 40
   D: 40 FORWARD 60
   D: 50 CENTRE
   D: END
HAND DEFINED
W: HAND
```

Irene changed the last two commands of 'SHAPE' and copied it as a new procedure named 'HAND'. Her substitution of the single command 'CENTRE' superficially solved the problem but anchored the procedure to a single position on the screen, which caused problems later. Since 'HAND' then seemed to work, Irene defined yet another procedure which copied all the commands of 'HAND' and added a few extra.

```
W: DEFINE "FINF
   D: 10 LEFT 40
   D: 20 FORWARD 60
   D: 30 LEFT 40
   D: 40 FORWARD 60
   D: 50 CENTRE
   D: 60 RIGHT 40
   D: 70 FORWARD 60
   D: 80 RIGHT 40
   D: 90 FORWARD 60
   D: 100 CENTRE
   D: END
FINF DEFINED
```

Irene then changed her strategy. She debugged 'FINF' by editing its definition and did not define any further redundant procedures in order to draw her diamond. This method did not produce deeply nested procedures.

Incremental

Incremental refinement was similar to linear refinement but produced much more deeply nested structures. It did not involve the definition of redundant procedures. In this strategy each procedure definition marked a stage in the process of solving the problem and did not correspond to any identifiable sub-part of the decomposed problem. Instead of inefficiently copying commands into a new procedure, as Irene had done, this strategy used the latest procedure as the first command in a new super-procedure.

This strategy was used by Jane to fill the display screen with

tessellated octogons. Her strategy had little overall plan but grew the picture incrementally using detailed local analysis. Jane planned a small part of the tessellation by defining, testing and debugging a procedure. When she was satisfied with it, a call to this procedure was included as the first command in a new procedure which drew a little more of the tessellation.

Four stages only from the process of drawing this tessellation are represented below.

(i) First Stage

Procedure 'BEE' was defined as follows, where 'PANSY' was a non state-transparent octogon. This procedure drew the shape given in figure 5.4

```
DEFINE "BEE
10 REPEAT 4 PANSY
20 LEFT 135
30 FORWARD 50
40 LEFT 45
50 FORWARD 50
60 RIGHT 90
70 REPEAT 2 PANSY
END
```

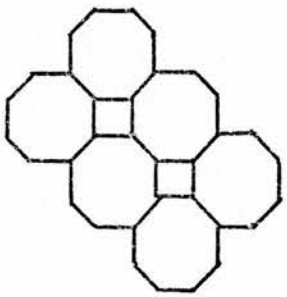


Figure 5.4.

(ii) Second Stage

A new super-procedure was then defined and debugged in which a call to 'BEE' was the first command.

```
DEFINE "SUPERBEE
10 BEE
20 LEFT 135
30 FORWARD 50
40 RIGHT 45
50 FORWARD 50
60 RIGHT 45
70 FORWARD 50
80 LEFT 45
90 PANSY
END
```

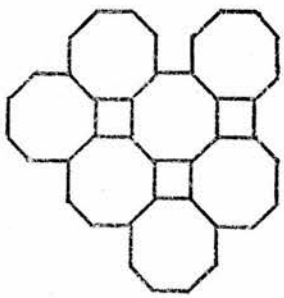


Figure 5.5.

This procedure drew the picture given in figure 5.5.

(iii) Third Stage

A new procedure 'SPIDER' was defined in which a call to 'SUPERBEE' was the first command. Procedure 'MAZE' was a sub-procedure which had been defined earlier. 'SPIDER' drew the picture given as figure 5.6.

```
DEFINE "SPIDER
10 SUPERBEE
20 MAZE
30 RIGHT 45
40 PANSY
END
```

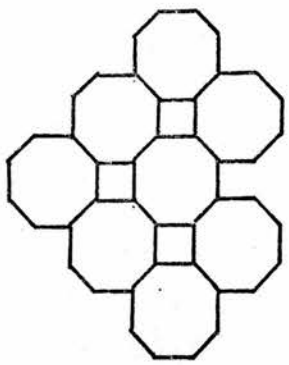


Figure 5.6.

(iv) Fourth Stage

Again a new procedure 'SUPERSPIDER' was built in which the 'SPIDER' was the first command.

```
DEFINE "SUPERSPIDER
10 SPIDER
20 WALK 50
30 PANSY
END
```

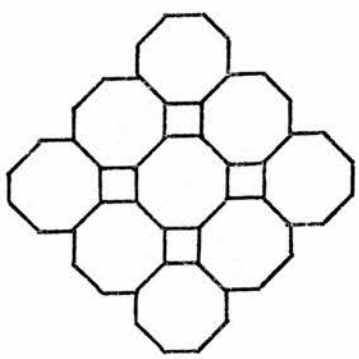


Figure 5.7.

The sub-procedure 'WALK' was also developed and acted as a set-up step between 'SPIDER' and 'PANSY'. 'SUPERSPIDER' drew the picture in figure 5.7.

Exploratory

Many of the programs written by the students were 'exploratory'. The students were not attempting to produce a given drawing but were

experimenting by seeing what drawing would be produced by their commands. Students soon learned that, by repeating the execution of a drawing procedure a number of times, they could produce pleasing pictures. If the procedure was not angle-transparent (i.e. left the turtle with a different heading at the end compared to the start) then pictures with rotational symmetry could be produced. This required that the procedure execution be repeated a sufficient number of times to bring the turtle back to its original heading. The correct number of executions could be determined by analysis of the procedure but was usually established by the students using trial and error methods.

Limited top-down

As students became more experienced, their strategy moved a little closer to a top-down approach. This required the programmer to define the main super-procedure first which will contain calls to sub-procedures, which are as yet undefined. In this way, the programmer gradually decomposes the problem into smaller sub-problems. But it is necessary to distinguish the process of decomposing a problem in this manner from a program which is organised top-down. Some of the students' programs had a hierarchic top-down structure, but were not written in that order. Students were usually very unwilling to include a call to an undefined sub-procedure in a definition unless they were extremely confident that they could easily define that sub-procedure. Many students, in group 1, found a programming exercise in one of the teaching notes uncongenial and the wrong way round. This described a program to draw a symmetric pattern made from rotated squares. In the example, the super-procedure was given first and the basic sub-procedure to

draw a side and corner of the square given last.

Students did become more adept at decomposing drawing problems into sub-problems but they preferred to work on the sub-problems first and when these were sorted out, they assembled them together.

5.3 AN ANNOTATED PROTOCOL

The following protocol exemplifies the difficulty which students encountered when they attempted to write programs of any degree of complexity. It illustrates the conditions under which the students worked and indicates the teaching methods adopted by the author. It puts the earlier analyses into a working context.

The student, Fiona, attempted to write a suite of procedures which would add, subtract, multiply and divide fractions. An example of her misunderstanding fractions was given in chapter 4. She muddled the divisive and multiplicative aspects of fractions in her attempt to show children how to change quarters into twelvths.

In the programming project the fractions were represented as two-element lists e.g. [3 4]. Fiona found this project difficult and it took her a large amount of time.

The problem given to Fiona was as follows:-

"To represent a fraction (a single entity) to LOGO the easiest way is as a two element list e.g.
3/4 can be written as [3 4]
5/6 can be written as [5 6]
Any fraction is just one member of a whole set of equivalent fractions. Can you write a procedure named EQUIVER which takes a fraction represented as a list and prints out all the equivalent fractions?
Write a procedure named NUMERATOR and another named DENOMINATOR which each take a fraction such as [7 8] as input and give as a result the appropriate number.
Arithmetic operations such as SUM or DIFFERENCE are only designed to work with natural numbers.
e.g. 1: PRINT SUM 4 6
10
Can you extend SUM by writing a new procedure FRACSUM which will operate on our new fraction numbers.

e.g. 1: PRINT FRACSUM [1 2] [1 4]
[3 4]

What about the other arithmetic operations, can you extend those also?"

Fiona's work on this problem extended over 8 programming sessions. Some of her coding difficulties and mistakes were caused by her transfer from the second to the third implementation of LOGO. She was also slowed by the unreliability of the two implementations. More importantly, she had much difficulty in decomposing the problems satisfactorily and in writing syntactically correct commands.

First session

At the very end of this session, Fiona attempted a definition of the procedure to print out equivalent fractions.

```
1: TO EQUIVER :NUM :START
&: 10 PRINT LIST PRODUCT FIRST :NUM :START PRODUCT LAST :NUM
 :START
```

Unfortunately the session terminated with a computer failure before she could complete the definition. Her procedure was correct as far as it went. The value of the argument 'NUM' was expected to be a two element list such as [4 5]. The value of the argument 'START' was expected to be an integer. Her procedure would have generated an equivalent fraction by multiplying the denominator and the numerator of the given fraction by the given integer.

Second session

session time: 1hr 35mins

commands: 57

error messages: 2

Due to the crash in the last session, Fiona had been unable to store her procedure and so she redefined it. She added a recursive call which incremented the value of the integer. But she left out

the second call on 'PRODUCT' in line 10 (see above) and so the procedure generated an error message when run.

```
1: TO EQUIVER :START :LIST
&: 10 PRINT LIST PRODUCT :START FIRST :LIST :START LAST :LIST
&: 20 EQUIVER SUM :START 1 :LIST
&: END
EQUIVER DEFINED
1: EQUIVER 1 [1 3]
[1 1]
LAST :LIST IS EXTRA. I IGNORED IT
IN EQUIVER
  10 PRINT LIST PRODUCT :START FIRST :LIST :START LAST :LIST
```

It is likely that Fiona would have been helped if she had used parentheses to delimit expressions. This was a fault of the teaching rather than the implementation. Again there was a computer failure as she tried to edit the procedure. After help with parsing the incorrect line, Fiona redefined the procedure when the system was ready. This time she omitted a ':' from in front of her argument name 'LIST'. This caused an error when the procedure was run, which she edited to produce a working version of 'EQUIVER'. She tried it out once and then stored it.

Fiona started on the problem of defining a procedure to add two fractions together. As suggested in the problem, she defined two utility functions to retrieve the numerator and denominator of a fraction. She did this correctly and tested each procedure once.

```
1: TO NUMERATO :LIST
&: 10 OUTPUT FIRST :LIST
&: END
NUMERATO DEFINED
1: PRINT NUMERATOR [4 12]
   4
1: TO DENOMINATOR :LIST
&: 10 OUTPUT LAST :LIST
&: END
DENOMINA DEFINED
1: PRINT DENOMINA [1 100]
   100
```

This implementation of LOGO truncated all procedure names to eight characters so 'DENOMINATOR' was equivalent to 'DENOMINA'. Fiona stored the two procedures and then decided that a further utility would be needed, which had not been suggested in the problem.

```
1: TO LCM :LISTA :LISTB
&: 10 OUTPUT QUOTIENT LAST :LISTA LAST :LISTB
&: END
```

Fiona was attempting to write a procedure to calculate the common denominator but divided rather than multiplied the individual denominators. She did not use the utilities, 'DENOMINATOR' and 'NUMERATO' in this definition. Having tried out 'LCM', she spotted and corrected the division mistake and stored a correct version of the procedure. This mistake is reminiscent of her classroom error where she muddled multiplication and division (see chapter 4).

Fiona then defined a procedure to add two fractions which used 'LCM' as a sub-procedure. The long prefix commands in this procedure are extremely hard to read and ought to have been parenthesised.

```
1: TO FRACSUM :LISTA :LISTB
&: 10 OUTPUT LIST SUM QUOTIENT LCM :LISTA  :LISTB FIRST :LISTA
   QUOTIENT LCM :LISTA :LISTB FIRST :LISTB LCM :LISTA :LISTB
&: END
FRACSUM DEFINED
1: PRINT FRACSUM [1 5] [1 3]
[30 15]
```

The procedure made the following incorrect calculation for the new numerator where the two fractions to be added are written as 'a/b' and 'c/d'.

a/b + c/d = (bd/a + bd/c)/bd

This is equivalent to asserting that

a/b + c/d = 1/a + 1/b

instead of

a/b + c/d = (a*d + c*b)/bd

Fiona was advised to put her procedure into a more readable form and reminded about assignment which could be used for this purpose. Fiona redefined the procedure.

```
1: EDIT FRACSUM
&: 10 MAKE "CALC LCM :LISTA :LISTB
&: 20 MAKE "BITA QUOTIENT VALUE "CALC FIRST :LISTA
&: 30 MAKE "BITB QUOTIENT VALUE "CALC FIRST :LISTB
&: 40 OUTPUT LIST SUM VALUE "BITA VALUE "BITB VALUE "CALC
&: END
FRACSUM DEFINED
1: PRINT FRACSUM [1 4] [1 3]
[24 12]
```

This edited version of the procedure contained the same incorrect calculation as before. Fiona stored the procedure and ended the session at this point.

Third session

session time: 2hrs 5mins

commnds: 45

error messages: 6

In this session the third implementation of LOGO was used. Fiona redefined her sub-procedure 'LCM' making the same division mistake she had made previously (and corrected). She made a number of syntactic mistakes due to the differences between the two implementations of LOGO. She then defined a new correct version of the fraction adding procedure, again using assignments to make it more readable. Although the procedure was correct, it calculated its result in an inefficient manner as follows:-

a/b +c/d = ((a*bd)/b + (c*bd)/d)/bd

instead of

a/b + c/d = (a*d + c*b)/bd

Fiona tried out the procedure twice and found that it worked correctly. She did not attempt to simplify it. More attempt should,

perhaps, have been made by the author to encourage Fiona to test her procedures more rigorously and to look for ways of improving them. She then experimented with the primitives 'REMAINDER' and 'QUOTIENT' as a preliminary step towards writing a procedure which would reduce any fraction to its lowest terms. There was a computer failure. When the system was restored, Fiona defined her cancelling procedure 'CANCEL'.

```
W: TO "CANCEL "LIST
   D: 10 MAKE "NUM FIRST :LIST
   D: 20 MAKE "DEN LAST :LIST
   D: 30 MAKE "REM REMAINDER VALUE "DEN VALUE "NUM
   D: 40 TEST GREATERQ VALUE "REM 0
   D: 50 MAKE "ANS QUOTIENT VALUE "REM 0
   D: 60 IFFALSE OUTPUT LIST VALUE "ANS VALUE "DEN
   D: END
```

Fiona stored the procedure and terminated the session without testing it. Her conception of the cancelling problem was to define a function, which given a fraction as input, would return a simplified fraction as output. The function would deal separately with the different classes of fraction which it might be given. The procedure, as defined, could only deal with the case where the denominator was a multiple of the numerator, i.e. it attempted to transform [3 12] into [1 4]. Fiona was surprised how hard the cancelling problem seemed to be turning out.

From this point on, Fiona used the linear refinement strategy to add more cases to the basic procedure. This strategy, which can be applied successfully (though inefficiently) to drawing problems is less successful in this type of problem.

With more experience, or with direct help from the author, Fiona might have realised that the problem she was addressing decomposed more neatly into a) finding the highest common factor of the

numerator and the denominator, and b) dividing this highest common factor into the numerator and the denominator.

Fourth session

session time: 1hr 40mins

commands: 22

error messages: 3

This and subsequent sessions (except no. 7) continued to use the third implementation of LOGO. Fiona retrieved the stored version of her procedure 'CANCEL' and tested it. She found that it incorrectly transformed [3 12] into [4 12] instead of into [1 4]. She edited the procedure, but it then transformed [3 12] into [1 12]. She edited it again and this time it appeared to work correctly. Her analysis of the problem had revealed that prime numbers were involved in cancelling. She wanted a procedure which could generate primes, but the author suggested that a list of primes might be a better, preliminary way to tackle the problem.

Again a computer failure interrupted the session. Afterwards, Fiona was helped to modify the title of her procedure to include a second argument named 'PRIMES', which would allow the procedure to deal with further fraction cases. Fiona terminated the session without testing or storing this version of the procedure.

Fifth session

session time: 2hrs

commands: 33

error messages: 2

At the start of the session, Fiona retrieved the stored version of CANCEL (which was not the most recently defined) and tested it. she found that it transformed [2 4] into [2 4]. Fiona made two

changes to the procedure, and partially repeated the work of the previous session. She changed the line which computed the returned value and she added a second argument named 'START' to the title. When trying the edited version, she forgot that it now needed two argument values, but correctly interpreted the resultant error message. She ran the procedure a second time by supplying a second dummy argument value of 0. The procedure made no use of this value.

```
W: PRINT CANCEL [2 4]
LOGO CANNOT FULLY EXECUTE THAT COMMAND, BECAUSE
THERE ARE NOT ENOUGH INPUTS FOR PROCEDURE CANCEL
W: PRINT CANCEL [2 4] 0
  [1 0]
```

Fiona then edited the command which computed the returned value and tested the procedure again.

```
W: PRINT CANCEL [2 4] 0
  [1 2]
W: PRINT CANCEL [3 15] 0
  [0 3]
W: PRINT CANCEL [3 9] 0
  [1 3]
```

The procedure behaved strangely, since it worked in two cases out of the three. Fiona worried (correctly) that her planning strategy was generating a larger and larger procedure which needed restructuring. But she did not know how to restructure it. The author suggested that she change from the 'TEST, IFTRUE, IFFALSE' form of conditional to the 'IF, THEN, ELSE' form to clarify the existing procedure structure.

Fiona edited the procedure again. She changed the name of the dummy argument from 'START' to 'PRIMES' and incorporated a list of primes into the procedure. This was accomplished by assigning a list of primes to the argument name (which would have overwritten any argument value!). She also gave the procedure a recursive structure

and added a stop rule which examined the numerator of the fraction.

```
W: EDIT "CANCEL
   D: 70
   D: 70
   D: 70
   D: RETITLE "CANCEL "LIST "PRIMES
   D: 6 IF EQUALQ FIRST "LIST 1 THEN OUTPUT "LIST
   D: 8 MAKE "PRIMES [2 3 5 7 11 13 17 19 23 29 31 41 43 47 53
    59 67 71 73 79 83 89 97]
   D: 70 IFTRUE IF EQUALQ REMAINDER QUOTIENT VALUE "NUM FIRST
    "PRIMES 0 THEN TEST EQUALQ REMAINDER QUOTIENT VALUE "DEN
    FIRST "PRIMES
   D: 80 IFTRUE MAKE "LIST QUOTIENT VALUE "NUM FIRST "PRIMES
    QUOTIENT VALUE "DEN FIRST "PRIMES
   D: 90 IFTRUE CANCEL "LIST "PRIMES
   D: END
```

These edits are extremely hard to read because of the lack of parentheses and have a number of mistakes. For example, argument names are used instead of values in several places. For example, line 90 should contain 'VALUE "LIST' and 'VALUE "PRIMES' instead of '"LIST' and '"PRIMES'. Her intention was for the recursion to terminate when the numerator was 1 (line 6) and for the fraction to be cancelled by the first of the prime numbers in the list, if that prime number was a factor (lines 70 and 80). Her recursive call did not take care of the case where the first of the prime numbers was not a factor. No provision was made to work down the list of prime numbers. Fiona asked if she could store a name/value pair in 'permanent memory' without having to imbed an assignment in a procedure. She was told that this was not possible. She tried out the new version of 'CANCEL'. Surprisingly it did not generate an error message. Her command muddled names and values since she supplied the literal word "PRIMES as the value of the second argument instead of a list of prime numbers.

```
W: PRINT CANCEL [4 12] "PRIMES
  [1 4]
```

She was pleased with the result and believed that the procedure had worked correctly. She failed to notice that the result should have been [1 3] rather than [1 4]. She terminated the session, having stored this version of 'CANCEL'.

Sixth session

session time: 1hr 45mins

commands: 51

error messages: 7

At the beginning of the session, Fiona retrieved and listed the stored version of 'CANCEL'. It was now a baroque procedure of eleven lines. Fiona spent the major part of this session defining and debugging a procedure 'MEMBERQ'. This was a predicate which tested whether an item was contained in a list. It was to be used in testing whether the numerator and denominator were both primes and so inhibit attempts to cancel the fraction when cancelling would not have been possible. She made many syntactic mistakes in her attempts to define 'MEMBERQ'. She entered the title incorrectly and she missed a 'THEN' after an 'IF', possibly by muddling the two forms of conditional. She also used the names of the arguments instead of their values (as she had previously). This produced non-terminating recursion which crashed the system. Her muddle between names and values may have been partly caused by the changes in implementation. In implementations 1 and 2, an argument name was given in the title of a procedure as ':ARG', say. Its value was retrieved in the body of the procedure in the same form, ':ARG'. In implementation 3, the argument was given in the title as '"ARG' and its value retrieved in a different form, either as 'VALUE "ARG' or as ':ARG'.

After help from the author, Fiona defined 'MEMBERQ' correctly

and tested it. It was then incorporated into an edited version of 'CANCEL' which attempted to deal with a wider variety of fractions. When this edited version of 'CANCEL' was tested it produced an error message due to an attempt to divide by zero. Fiona stored the procedure and terminated the session without further attempts to debug the procedure. She was still using the literal '"PRIMES' as an argument value for her procedure.

Her procedure was now as follows, though she did not list it in its entirety. The probable intention of each line is given in lower case. The procedure has been parenthesised by the author to make it more readable. Fiona's version contained no parentheses.

```
DEFINE "CANCEL "LIST "PRIMES
4 IF (BOTH (MEMBERQ (FIRST :LIST)(:PRIMES)) (MEMBERQ (LAST :LIST)
:PRIMES)) THEN OUTPUT (LIST (FIRST :LIST)(LAST :LIST))
```

If numerator and denominator are both primes, return the fraction uncancelled. This misses the case where they are both primes but equal, e.g. [7 7]. Her construction (LIST (FIRST.....:LIST)) is equivalent to (:LIST).

```
6 IF (EQUALQ (FIRST :LIST) 1) THEN OUTPUT :LIST
```

If the numerator is 1, return the fraction.

```
8 MAKE "PRIMES [2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67
71 73 79 85 89 97]
```

Assign a list of prime numbers to variable 'PRIMES'.

```
10 MAKE "NUM FIRST :LIST
```

Calculate the numerator.

```
20 MAKE "DEN LAST :LIST
```

Calculate the denominator.

```
30 MAKE "REM (REMAINDER (VALUE "DEN)(VALUE "NUM))
```

Integer divide denominator by numerator and assign remainder.

```
40 TEST (GREATERQ (VALUE "REM) 0)
```

Set a flag if the remainder is greater than zero.

```
50 MAKE "ANS (QUOTIENT (VALUE "DEN)(VALUE "NUM))
```

Divide denominator by numerator and assign the quotient.

```
60 IFFALSE OUTPUT (LIST (QUOTIENT (VALUE "NUM)(VALUE "ANS)) (QUOTIENT
(VALUE "DEN) (VALUE "ANS)))
```

If the flag is set false, return a fraction given as [a b] transformed into [a/(b/a) b/(b/a)].

```
70 IFTRUE IF (EQUALQ (REMAINDER (QUOTIENT :NUM (FIRST :PRIMES)) 0))
THEN TEST (EQUALQ (REMAINDER (QUOTIENT :DEN (FIRST :PRIMES)) 0))
```

Both the consequent and antecedent contain errors. In each case the '0' is intended as the second argument to 'EQUALQ' but is the second argument to 'REMAINDER' (hence the earlier error message about dividing by zero). The intention is to test whether numerator and denominator are multiples of the first of the prime numbers, setting the flag if this is the case. The IFTRUE is redundant, since the procedure would terminate in line 60 if the flag was set false.

```
80 IFTRUE MAKE "LIST LIST (QUOTIENT (VALUE "NUM)(FIRST :PRIMES))
(QUOTIENT (:DEN (FIRST :PRIMES)))
```

Cancel numerator and denominator by the first prime number and assign to the variable holding the fraction.

```
90 IFTRUE CANCEL :LIST :PRIMES
```

Recursively call 'CANCEL' with the partially cancelled fraction.

```
100 IFFALSE CANCEL :LIST (BUTFIRST :PRIMES)
```

Recursively call 'CANCEL' using the remainder of the list of primes but the same fraction as given. The use of 'TEST' mixed with 'IF' makes analysis of possible flows of control through this procedure complex.

Seventh session

session time: 2hrs 15mins

commands: 77

error messages: 13

Unfortunately implementation 2 had to be used for this session because implementation 3 was not available. Firstly, Fiona redefined 'CANCEL' in the appropriate format but with the same code as in the last session. She introduced a number of mistypings, e.g. 'VALE'

for 'VALUE' and 'T EST' for 'TEST', but successfully changed the format. She also omitted a procedure call from line 70. Having redefined 'MEMBERQ', she stored the two procedures and tested 'CANCEL'.

```
1: PRINT CANCEL [15 20] :PRIMES
```

This command produced an error message because no value had been assigned to the variable ':PRIMES'. It should have had a list of prime numbers as its value. Fiona tried to avoid the problem by redefining 'CANCEL' so that it did not have two arguments. She tried running 'CANCEL' again but the fault persisted because line 4 of the procedure still referred to the argument name 'PRIMES'. With help from the author, Fiona moved the assignment (line 8) of 'PRIMES' to the first line of the procedure.

This time when 'CANCEL' was tested it resulted in a stack overflow because MEMBERQ had recursed more deeply than the system could support. This was fixed, at the author's suggestion, by using a shorter list of prime numbers. Some time was spent tracking down and eliminating the typing mistakes which had been introduced earlier.

The author then suggested that her procedure be slightly restructured into a super-procedure and a sub-procedure. The new super-procedure was as follows:-

```
1: TO CANCEL :LIST
&: 10 MAKE "PRIMES [2 3 5 7 11 13 17 19 23 29 31]
&: 20 CANCLER :LIST :PRIMES
&: END
```

This super-procedure assigned a list of primes to a global variable named 'PRIMES' and then called a sub-procedure, 'CANCLER' which did the cancelling work (Fiona's old procedure). Fiona carried out the

restructuring and tried out the new procedure.

```
1: PRINT CANCEL [15 20]
[ ]
```

This empty list output caused Fiona some wry amusement.

Eighth session

session time: 2hrs

commands: 52

error messages: 7

The author transferred copies of the procedures produced in the last session to implementation 3 (in a suitably amended format). Their names were given the prefix 'BEN' so that they would not overwrite Fiona's existing procedures. This would give her the option of deciding which procedures she wished to use. She opted to use the transferred procedure but had to edit them. This was because they needed an explicit command to return a value (it had happened by default in implementation 2).

Fiona tried out the procedures again and was very pleased when they appeared to work, exclaiming "Wow" when the result was printed.

```
W: PRINT BENCANCEL [4 6]
[2 3]
```

Her next test was not so successful:

```
W: PRINT BENCANCEL [16 20]
[1 1]
```

and she described the machine vividly, "Its a horrid grudging thing."

The author showed her how she could trace the execution of the procedure. This helped her understand how her procedure was working. She identified line 80 as the cause of the trouble and commented accurately:

"Its all my ruddy IFTRUES"

In trying to sort out the bug she had a flash of inspiration:

"The great light is, if the denominator is a prime number, em, you can't cancel any more."

The author pointed out a counter-example to this, [13 13]. Fiona decided to add commands to the beginning of her procedure to deal with cases where the denominator was a prime. She was able to give a verbal description of the action of the procedure and explained how it dealt with different fraction cases.

```
W: EDIT "BENCANCLER
   D: 4 IF MEMBERQ LAST :LIST :PRIMES THEN OUTPUT :LIST
   D: END
```

The procedure was tested, but it incorrectly transformed [15 20] into [3 5]. Since tracing was switched on, Fiona was able to see the cause of the trouble. This was that [15 20] was transformed into [7 10] and then into [3 5] using integer division. This depressed her and she said she felt like giving up. The author encouraged her, and she made further edits and tried the the procedure again. As execution proceeded (with tracing on), Fiona correctly anticipated that an error was going to occur because the prime number list was being exhausted and there was no stop rule to deal with it. At the moment at which the error eventually occurred, her procedure had correctly transformed [15 20] into [3 4].

Fiona stored her procedures and terminated the session without further debugging. She abandoned the problem and did not return to it again.

Fiona spent much time and effort on this problem. Some of her classroom difficulties with fractions reappeared in her program as planning mistakes. Her main difficulty lay in formulating a clear plan for the cancelling algorithm which she was trying to program.

Further difficulties were caused by implementation changes and unreliability. Her failure to parenthesise expressions made her code hard to read and error prone. She muddled names and values and had decomposed the problem in a way that led to difficulties. Her analysis based on 'cases' using linear refinement produced an unwieldy procedure which was hard to debug. Although she realised this, she could not see how to decompose the problem into a more amenable form.

5.4 SUMMARY

It was found that the students faced many difficulties in coding, planning and debugging programs. Programs of any complexity took many sessions to write and the students needed much help. This severely restricted the kind of programming work which it was feasible for the students to undertake given their limited time. Early emphasis on drawing gave the students a motivating introduction to programming but encouraged them to use bottom-up problem-solving strategies. This could possibly be counteracted by introducing programming with much more emphasis on planning and analysis. It was also found that although students did make mistakes in arithmetic computation and misunderstood some rules of computation, their greater need was to understand the meaning of these rules. This cast doubt on the wisdom of asking students to rewrite rules, which they already knew, using a new formalism (LOGO) in the hope that, by changing formalism, the meaning of those rules would emerge.

CHAPTER 6

JANE:

LEARNING MATHEMATICS THROUGH PROGRAMMING

This chapter and the next examine how the students learned mathematics through programming. Detailed case studies of the work of Jane, Irene and Mary are presented. These three students brought contrasting degrees of competence and self-confidence to their programming work and reacted to it in very different ways. This chapter deals with Jane.

6.1 ADVANTAGES AND DISADVANTAGES

Learning mathematics through programming had both advantages and disadvantages, which are summarised in a table (see figure 6.1). These attributes are now described against the background of the claims commonly made for this approach to learning mathematics (reviewed in chapter 2), the mathematical difficulties of the students and their difficulties in learning to program.

The work given to the students did not attempt to illustrate the idea of mathematical rigour (e.g. as applied in proofs) but succeeded in demonstrating the value of the weaker idea of 'explicitness'. Students realised that the lack of ambiguity in the programming language enabled them to communicate their intentions to the computer exactly. But this was often a frustrating experience. Some students did not find the required precision of expression congenial and compared the effort involved unfavourably with the result (e.g. an hour or so spent drawing a simple house outline).

Programming provided extensive opportunities for the students to

engage in active mathematical exploration, particularly of Turtle Geometry. Mary made several personal mathematical discoveries by investigating the mathematics underlying her programming. Different primitives were needed for other domains. These turned out to be too complex for the students to construct for themselves out of LOGO and were provided for them. This happened to an increasing extent towards the end of the study. It freed the students from the need to worry about tedious and often irrelevant programming detail and enabled them to concentrate on the mathematical properties of the given new primitives. For example, primitives were provided to illustrate the symmetry transformations of the rectangle and to give a visual interpretation of fraction operations based on ratios. In both cases the main programming effort concerned the method of mapping from the internal representation to the drawings on the display screen. This had scant mathematical value and would have been an inappropriate task for these students, even if they had the programming skill to undertake it. The students concentrated on the way the primitives behaved rather than on the way they were constructed. This did not mean that they did not think carefully about the behaviour. For instance, Jane found a mathematical bug in one of the primitives.

Programming illustrated a number of key concepts including function, algorithm, the state/transformation distinction and angle as rotation. These concepts were embodied in the structure of LOGO. Sometimes students explored a concept by observing how the primitive behaved (e.g. what it drew). By contrast, Mary was able to use the code of hypothetical procedures to learn about functions. But the availability of the computer tended to distort the students' work

towards what was programmable rather than what was mathematically most beneficial. For example, it was found in many cases that students knew certain algorithms (which were easy to program) but they did not understand the meaning of those algorithms or how they might justify and explain them to their pupils. This made the reproduction of the algorithms in a program a pointless activity.

The students had splendid opportunities for problem-solving. Mary and Jane used this to learn the value of the strategy of problem-decomposition. But the ease of access to the machine encouraged trial and error or bottom-up methods rather than planning and analysis. For instance, certain figures were drawn without reference to their overall geometric properties. Instead the student used bottom-up 'incremental' strategies which depended only on detailed low-level analysis. Most students thought about their programming activity and the way they had learned to program. Mary and Jane were able to make some analogies between their experience of learning programming and children's experience of learning mathematics.

Most students enjoyed learning to program using Turtle Geometry but found certain concepts hard to grasp e.g. variables. The bottom-up problem solving stategies which the students evolved were not appropriate for other classes of problem e.g. symbol manipulation problems and decomposition into functions. Some students, notably Irene, found programming difficult and just as frightening and unpleasant as mathematics had been.

Some small improvements in attitude to mathematics were observed. These were linked to the particular mathematical topics which had been explored. The students' overall self-confidence and

attitude to mathematics was not changed (and seemed very hard to change). Many of the students held the pragmatic view that their main task was to pass their Diploma and learn a number of 'recipes' for teaching mathematical topics. For them, learning programming in order to understand mathematics better seemed a circuitous route to more successful teaching. Differences of initial attitude were important. Mary seemed to believe that her mathematical difficulty consisted of ignorance or lack of understanding of specific topics. These she regarded as separate, solvable sub-problems which could be tackled individually. By contrast, Jane imbued her ignorance and lack of understanding with a general belief in her own mathematical incompetence. Although she became more confident about teaching specific topics, her overall attitude remained unchanged.

It was found that students needed explicit help in linking the mathematical content of their programming projects to their existing mathematical knowledge. For example, Jane still had difficulty understanding clockwise and anti-clockwise rotations on a protractor after extensive work in Turtle Geometry. Only after the author linked the turtle rotations specifically to the protractor did she see the connection.

The two major disadvantages of learning mathematics through programming were that programming itself was a complex skill for these students to learn and that it was all too easy to give the students problems to solve at the wrong level of representation. For example, some students were asked to study fractions by writing procedures to draw fraction pie-charts. This was a poorly chosen project because most of the students' efforts were directed at getting the drawing correct. This could be achieved (or could fail)

with little reference to fractions and did not help their understanding of fractions. In this project, and in another on vectors, the students were asked to give commands which drew representations of the given mathematical structures. This was not the same task as giving commands to manipulate an internal representation of those structures, where the manipulation had the side effect of producing a drawing. The latter task would have been more valuable because it involved understanding the structures themselves rather than the pictorial qualities of their representation. In general, it was not easy to design programming projects which helped the students deal effectively with their classroom difficulties. This was because the programming was too complex, or because the behaviour of the program or its code did not demonstrate the idea in question clearly.

The main advantages of programming were that it presented mathematics as an exploratory activity involving problem-solving rather than rote learning. It provided concrete illustrations of a number of abstract ideas. Finally, it gave the students the opportunity to be honest about their mathematical difficulties and to discover that they had the ability to overcome at least some of them.

ADVANTAGES	DISADVANTAGES
Emphasises importance of explicit language.	But causes frustration because making intentions clear can take a long time.
Provides opportunities for active mathematical exploration, e.g. Turtle Geometry.	But primitives for other domains are too complex for students to write for themselves. Students may work at the wrong level of representation.
Certain key concepts are well illustrated e.g. function algorithm, angle, state and transformation.	But availablity of the computer distorts syllabus towards what is easy to program rather than what the student needs. It is not easy to design programming projects which address the mathematical difficulties of students.
Splendid opportunities for problem-solving are available.	But ease of access to the machine encourages trial and error rather than analysis.
Learning programming through Turtle Geometry is fairly effective and fun.	But certain concepts still difficult e.g. variables. Strategies evolved in solving drawing problems inappropriate for other classes of problem. Some students find programming just as frightening as mathematics.
Successful completion of projects improves attitude to the topic studied.	But overall attitudes to mathematics hard to change Student teachers often more concerned with recipes for successful teaching than with understanding the topics they are to teach.

Figure 6.1. Advantages and Disadvantages of Learning Mathematics Through Programming.

We now present the first and most detailed of three case studies. These reveal the process by which the students learned mathematics and show how the different students benefited in different ways from the experience.

The evidence presented is of three kinds:-

(i) The protocol of the interactions at the computer terminal between the student and the computer, between the student and the author and between student and student. These consisted of listings of the programming work, work done by the student, notes made by the author and audio tape-recordings of tutorial dialogues.

(ii) The audio-recordings of the students' lessons and the recordings and notes of their reactions to hearing recordings of those lessons.

(iii) The students' written answers to the second and third questionnaires.

The case studies present only a small part of the data collected. Incidents were selected which exemplified the advantages and disadvantages of learning mathematics through programming and which typified students' reactions to the experience. Sessions which were concerned mainly with learning programming have not been included because the students' experiences of learning to program have already been described.

Jane spent about 84 hours in the programming classroom writing and debugging programs as well as discussing mathematics. This does not include the time spent discussing recordings of her lessons or the time spent in meetings at the College of Education. The number of hours Jane spent programming, month by month, is shown in figure 6.2.

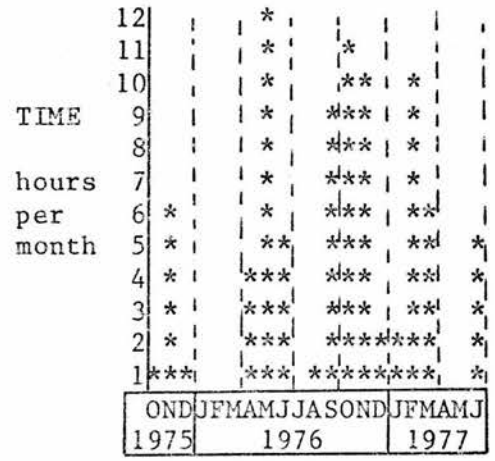


Figure 6.2. Jane's Programming Sessions.

In all Jane took part in the project over a period of 22 months. The three monthly vertical divisions in the chart correspond roughly to College of Education terms.

Jane's work will be presented by concentrating on the evolution of her understanding of particular topics, such as angles. In this way it is intended to trace the course of the interaction between her programming and her mathematics.

The next three sections analyse her work in the areas of geometry, algebra, and number. A final section summarises her work in relation to the framework set out in section 6.1.

6.2 GEOMETRY

In chapter 4 we saw that Jane had a number of misunderstandings about geometry both at the start of this study and during the course of it e.g. use of a protractor. This section describes the programming work she undertook and the effects it had on her understanding of such concepts as angle. Angle as rotation is a central concept in Turtle Geometry. It will be shown how Jane's

attitude to, and skill in dealing with, angles improved as a result of her programming work. As her work progressed it will be shown how her attention shifted from the properties of particular angles, with which she had become familiar, to the more general properties of shapes, for example their symmetry and their total angle properties.

The importance of good teaching will also be seen, especially where opportunities were lost for exploiting Jane's programming work for mathematical purposes. Evidence will also be presented which suggests that despite her extensive angle work in the programming classroom, Jane failed in some respects to link it with her existing angle knowledge. Instances will also be given of how concentration on the programming aspects of a problem sometimes masked what was important mathematically.

Learning to visualise angles

The elements of programming were learnt, in the first term, by writing procedures to control the floor turtle or the simulated turtles in the display and graph-plotters. Jane succeeded in drawing a number of designs including regular polygons. This necessitated that she distinguish translations from rotations, and clockwise rotations from anti-clockwise rotations. In order to draw a given polygon it was necessary for her to know the values of its angles in degrees and to distinguish interior from exterior angles of the polygon. Figure 6.3a illustrates our use of the term 'exterior' angle of a polygon. Occasionally she failed to make this distinction and produced, for example, part of a regular hexagon instead of an equilateral triangle, see figure 6.3a. This was a common mistake among the students, probably accounted for by the misapplication of

the rule that 'the three angles of a triangle add up to 180 degrees'. It may also have been affected by drawing squares and rectangles whose interior and exterior angles are both 90 degrees (see figure 6.3b) and whose successful completion did not confront the student with this interior/exterior angle distinction.

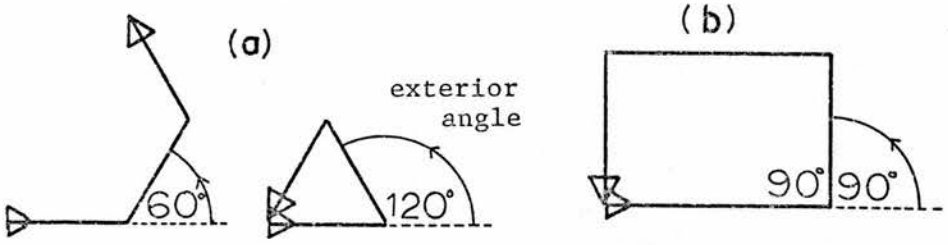


Figure 6.3. Interior and Exterior Angles.

When Jane returned to programming, after teaching practice, she made just the same mistake when drawing an equilateral triangle, as part of the problem of drawing a 'house'. Her initial diagnosis was that she had got the direction of the rotation wrong rather than its magnitude:-

"I've done it the wrong way. I said right instead of left"

"...instead of saying left sixty, I said right sixty"

The author asked whether a triangle could be drawn by turning right. Jane agreed that this was possible and then sorted out the difficulty on her own, as follows. She next tried a rotation of RIGHT 150, but changed this to LEFT 120 when she saw its effect. She was asked what the trouble with her procedure had been and replied:-

"Well, em, I should have done a hundred and twenty plus, a hundred and twenty round...you see six...I always get this wrong. I keep forgetting that the point's [the light-spot representing the turtle on the display] going that way and

therefore you have got to get that far round. I think it...I always think it's like a protractor...that...I was measuring sixty but we should have been six...ninety plus thirty, that's one hundred and twenty"

The value of programming in this context is clear. Jane was able to solve a problem herself because she was able to exploit the link between her explicit commands and the machine's visible reactions to those commands. The separation, in LOGO, between procedures for producing rotations and those for producing translations is valuable in this respect. However her explanation of her difficulty and its solution, particularly the phrase 'ninety plus thirty', suggest that she was not using the idea that the interior and exterior angles are supplementary (add up to 180 degrees). She seems to have solved the problem by a process of successive refinement, after her initial attempt. This problem was to recur with her and other students. The ease of access of the machine on which experiments could be conducted tended to prevent the students conducting a rigorous analysis. Very often local problems could be solved by trying solutions which were gradually refined. However once the solution had been found, there was little incentive to examine it, or the process leading to it, to find other solutions.

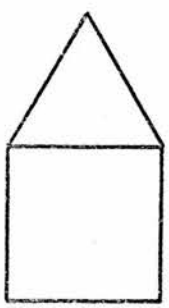


Figure 6.4. House.

Jane then spent some time fitting the triangle correctly on top of the square to finally complete her 'house', see figure 6.4. It took quite a lot of time and effort partly in dealing with procedure management (e.g. defining, editing, listing and saving procedures) and the syntax of the language and partly in working out how much to rotate the turtle between drawing the triangle and the square. Difficulties with the language were aggravated by the change to implementation 3. When it was done she explained:-

> "Oh, huh, its an awful lot of paper just to do that. I suppose you learn by...by making mistakes and having to put it all together again...I think you do. It really makes you think about it. I think what really stumps me is the fact that I can never remember which way it's [the simulated turtle] facing. Like to make the angles, I completely forget which way, like it is going that way and it isn't like a protractor and facing up the way."

Part of her difficulty was caused by the fact that the current heading of the simulated turtle on the display was not shown continuously. It was only possible to request that it be shown briefly. The author suggested that she did some more work with the floor turtle whose heading was easier to see.

Her phrase 'just to do that', that is draw a simple house, indicated that in terms of a product the session had been hard work for a small return. But she qualified this with her remarks about learning from mistakes which suggested that she saw value in the process. That is to say, the rigour demanded by the computer was seen as valuable because it forced her to formulate a completely explicit description of the 'house'. It also suggested that she was

looking on her mistakes constructively and was using the activity to 'think about thinking', though without using computational terms to describe her thinking. By virtue of this explicitness, she could debug her procedures. But as we have noted, there was little overall analysis of the problem. This disadvantage of programming might be diminished if students were directed through a more formal procedure planning stage than was the case in this study.

The author lost the opportunity for making Jane be more verbally explicit about the interior/exterior angle distinction. He just accepted her phrase 'it isn't like a protractor' which he understood to refer to the interior angle. It seems clear that whatever potential the programming activity has as a method of studying mathematics, skillful teaching still has a vital part to play. Here the explicit computational description of the square and triangle could well have been complemented by an overall analysis of the solution found and by an explicit English description of their properties and interaction. Just because Jane had succeeded in solving the current problem did not necessarily mean that she had understood her solution.

At the next session, she used the house procedure to draw a whole 'street' of 'houses'. When this was drawn on the display screen it came out upside down, because of the initial heading of the turtle at the start of the drawing. This could have been cured by initially rotating the turtle 180 degrees. Instead Jane wondered:-

"Would that be three...that would be right...that would be right round, er, wouldn't it."

The author said that turning the turtle right round would make no difference, but the problem was left hanging to be solved later.

Jane seems to have muddled turning something upside down, rotating it through 180 degrees, and turning something right round, rotating it through 360 degrees. Again a mathematically interesting point was passed over because, in some sense, the programming problem had already been solved. A 'house' had been drawn successfully. It was easier to turn the paper upside down with the house drawn on it than to worry about why the house came out upside down.

Some of the questions in questionnaire (2), administered that term, concerned the planning of drawing procedures. The students were asked how they would set about producing the pattern given in figure 6.5. Jane gave a clear English description, breaking down the problem into appropriate sub-problems, of which drawing a house shape was one. She correctly explained that each house would have to be rotated through 60 degrees in relation to its neighbour.

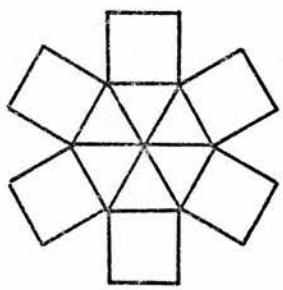


Figure 6.5. Rotated House.

Another question asked if she could see any use for her programming work in her future teaching. She replied:-

"I've a much better idea of angles degrees etc. than I had when I started--before I couldn't have attempted it without a lot of help."

An important phrase here is 'before I couldn't have attempted it'. It suggests a positive change of attitude about her ability to tackle this topic in the classroom which may be contrasted with her normal lack of mathematical self-confidence (see chapter 4).

At a later session that term Jane wanted to draw a rhombus which she called a diamond. At first she wrote a procedure in which all the angles were 45 degrees that produced a drawing as in figure 6.6a. She then changed all the angles to 135 degrees which produced a drawing as in figure 6.6b. This suggests that she was still confused about the interior/exterior angle distinction but knew they were supplementary. She sorted out the problem on her own to produce a non-state-transparent rhombus with angles of 135, 45 and 135 degrees as figure 6.6c. A state-transparent figure leaves the turtle with the same heading and position as at the start of the figure.

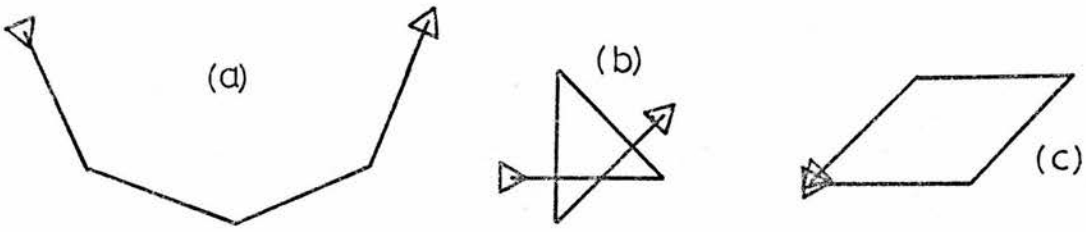


Figure 6.6. Drawing a Rhombus.

She tried to use this rhombus to draw a 'flower figure' as in figure 6.7, but found difficulty in aligning the turtle between each rhombus, no doubt because of the lack of state-transparency. She did not seem to have made an overall analysis of the problem, as she had done in her answer to the questionnaire question about rotated houses, and was proceeding bit by bit to build the flower.

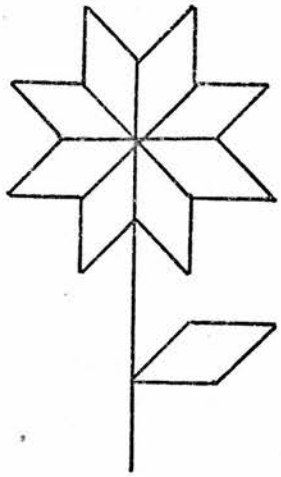


Figure 6.7. Flower.

Drawing Polygons

Later in the term Jane gained further experience of angle properties of polygons. The session was designed to enable Jane to explore angle properties of regular polygons and to consolidate previous work with user-defined procedures which took arguments. She was shown how to define a procedure of two arguments which could draw any regular polygon depending on the values of the arguments (one for the exterior angle, the other for the number of sides). She was introduced to the term 'regular' and after some initial work with convex polygons, it was suggested she try drawing 'stars'.

She tried to draw a star by choosing various argument values for the procedure but could not, at first, make the ends join up, see figure 6.8. She had not understood that the condition for this to happen was that the turtle rotate a whole number of complete revolutions in tracing the star. This is known as 'the total turtle trip theorem'.

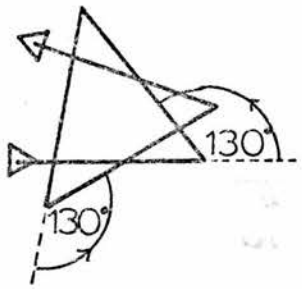


Figure 6.8. Failed Star.

Discussion revealed that she did understand the relation between the sharpness of the star's vertices and the angle argument of the procedure. That is that 150 degrees would produce a more sharply pointed star than 130 degrees. Eventually she was able to draw a five pointed star by trial and error. Her last two attempts used angles of 145 and 143 degrees. Within the limits of accuracy of the display these could not be distinguished from the star with the correct angle of 144 degrees. She was asked if she knew any rule which could help her to draw stars but she said not. She then spent some time modifying her polygon procedure, by adding an extra variable, so that polygons of any size could be drawn. She drew a number of decagons of different sizes and understood that these closed because she had specified a complete revolution in ten individual turns of 36 degrees each, making a total of 360 degrees in all.

She then returned to the problem of the stars, but could not see the relevance of the above condition of closure of the decagon, especially as the total angle turned in drawing her stars was much greater than 360 degrees. This difficulty was aggravated because she was certain that her five pointed star, using 143 degree angles, had in fact closed. Thus when she was asked to calculate its total angle

turned, she correctly calculated 5*143=715 degrees but saw no significance in this result.

The author asked her to imagine herself walking around the star, but in the end he specifically pointed out that exactly two complete revolutions closed the figure. The dialogue became very one sided, as the author explained to Jane. At first Jane thought that failure to close was because she had failed to arrange for her procedure to turn the final corner of the star, but this was not the case. She was near the truth because she was using the idea of state-transparency in her analysis of the problem, but was not using it correctly in this case. Eventually she understood, after further intervention by the author, that a five pointed star needed a total turn of 720 degrees. That is two complete revolutions made up of five rotations of 144 degrees. Jane explained her difficulty:-

> "I didn't realise that with a star. You know, I never thought in terms of going round several times. I was always thinking in terms of one revolution. Always it was three sixty."

Unfortunately the opportunity was not grasped by the author to suggest that she draw other stars and the session moved on to a new topic. This incident shows up one practical difficulty associated with the particular drawing devices in use. It also shows how both student and author were overconcerned with the product, that is in this case a star, so that insufficient time was spent on studying the general properties of stars. Perhaps if the author had not pushed Jane for an explanation of her first star, the total turtle trip theorem would have emerged more naturally from Jane's own exploration of the problem of drawing stars with different numbers of vertices. As it was, the author gave her 'the answer'. A further difficulty

was caused by the vagueness of the instructions, 'try to draw a star'. It would have been better to suggest she drew a number of stars possibly with specified numbers of vertices. At the end of the session, which had also included some work on functions, Jane was asked if she felt her programming work was helpful. She replied:-

> "Well, yes, I think so because it really makes you think about, you know, how, how you do these procedures...Yes, well, I think with the angles...I think it's, I think it's probably getting a bit easier now to think about angles. Whereas before it was just a, a haze. I didn't have a clue...I...usually I am absolutely terrified in case any of the children in the school had asked me about angles because I was just...you know, I would have to go away and really think about it and then come back. But if they want an answer right there and then, I just used to have to fob them off with something else."

Her fear of being found out not knowing the answer to a chid's question has already been commented on in chapter 4. She is still rather guarded in her evaluation, 'a bit easier now to think about angles'. Her statement suggests a change of attitude because she believed herself to be now more capable of tackling this topic in the classroom. This belief in her ability to cope mathematically is important for her success and happiness as a mathematics teacher. Of course mere belief is not enough, it must be under-pinned by understanding, but it remains an important factor.

Looking for Patterns

Jane continued to program during the summer vacation. This was partly because she enjoyed it and also because she did not want too

long a gap in her work. The last time this had occurred, during the previous teaching practice, she felt that she had had trouble revising programming.

One of the problems she worked on was to draw a 'tree' using an 'arrow' sub-procedure, see figure 6.9a. During the course of this problem, Jane explored how considerations of symmetry enabled her to deduce the angle properties of a figure. Her analysis was initiated as a way of reducing inefficient computation. She was trying to construct a mathematical rule for herself. She defined an arrow procedure with arguments which allowed her to vary the size of the arrow's arms and spine, as well as the inclination of the arms to the spine, see figure 6.9b. In the figure, the angles which affected the inclination are marked alpha and beta. She wished to draw an arrow whose arms were at equal inclinations. After a calculation, described presently, she tried angles of 135 and 215 degrees which gave her an arrow similar to that in figure 6.9b.

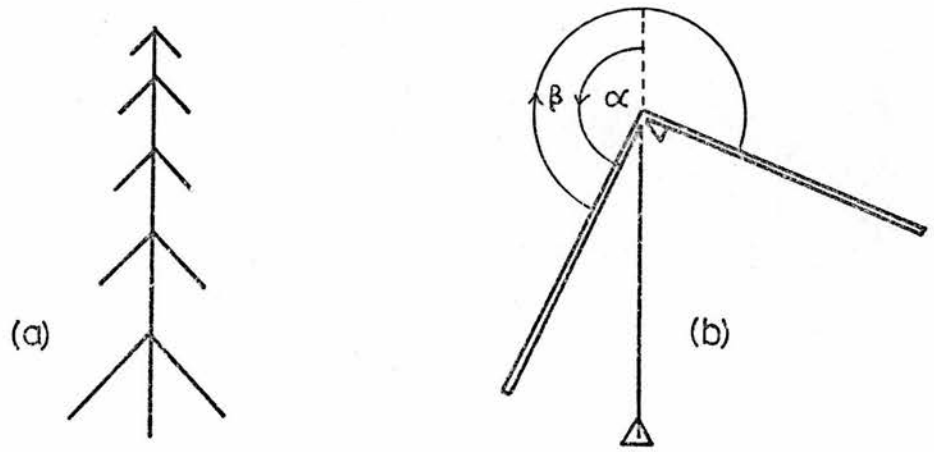


Figure 6.9. Tree and Arrow.

She then recalculated and tried 135 and 270 degrees which gave the required effect. She realised that the symmetry of the arrow would force a numerical relation on these two angle values but did not know

what that relation was.

She asked for help, explaining that she did not want to redo her (laborious) calculations each time she wanted to draw an arrow with arms at equal but new inclinations to the spine. She knew that having selected the value of one angle, then the value of the other angle was determined, but did not know how to find it. This provides an example of the generation of a search for a mathematical abstraction by the need to solve a concrete problem more efficiently. Strangely, Jane was not alerted to the relation (that one angle should be twice the other (see figure 6.9b) by the values 135 and 270 degrees. This time the author suggested that she solve this problem for herself.

By the next session, she had solved the problem and understood that one angle had to be twice the other. She explained that previously she had calculated the values by breaking up the rotation into parts (see figure 6.10) and then adding those parts. It had been a mistake in this addition which had made her use 135 and 215 degrees instead of 135 and 270 degrees. She also explained that she had:-

> "looked and looked [at 135 and 270] and thought there was no relationship at all...[because]...I said twice one thirty five is three seventy."

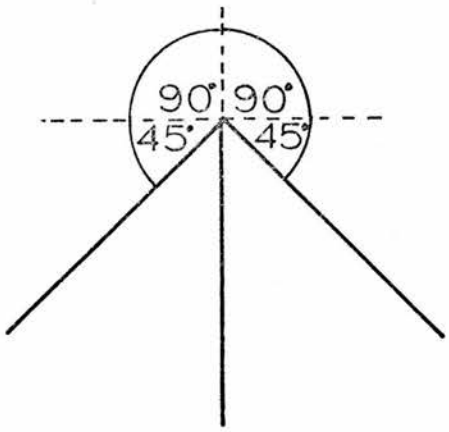


Figure 6.10. Arms of Arrow.

This suggests that she might well have suspected what the relationship was but was misled by an arithmetic error. Similarly in the incident of drawing the star, earlier, she did not notice that 715 was approximately equal to 720 (2*360). Having successfully mastered the arrow problem she went on to draw a 'wood' full of 'trees' made up of 'arrows'. This incident supports the claim that programming provides an activity in which mathematical questions can be tackled. Here the mathematics was employed by Jane to make a problem solution more elegant. But it also indicates that the activity should be assisted by sympathetic teaching.

In a later session Jane explored a number of different recursive procedures including ones which printed out a Fibonacci series (see section 6.4) and another which generated a number of spirals. The spirals procedure was a generalisation of the polygon procedure she had already worked with. The spiral was produced by drawing line segments of increasing length whose angle of inclination to each other was fixed, see figure 6.11. This gave her the opportunity to explore the properties of obtuse and reflex angles.

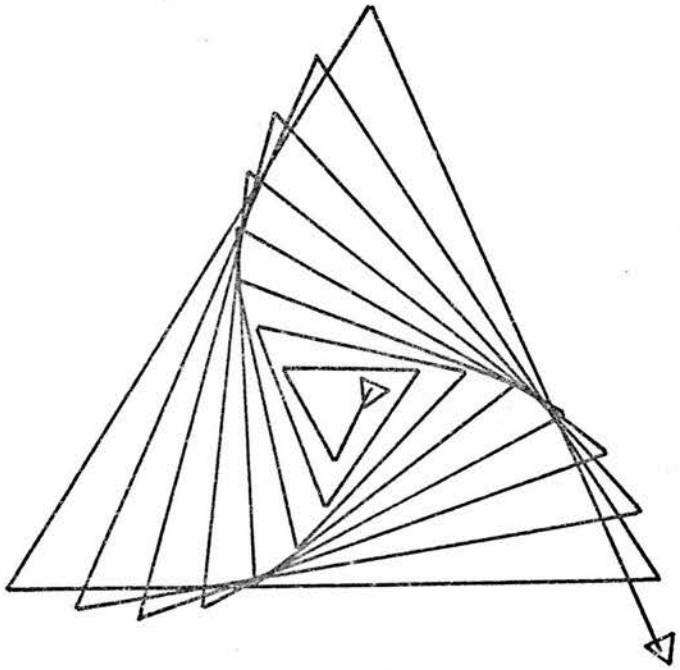


Figure 6.11. Spiral.

She worked on the problem of reproducing the sharply pointed spiral of figure 6.12 which was also on the wall of the programmimng classroom.

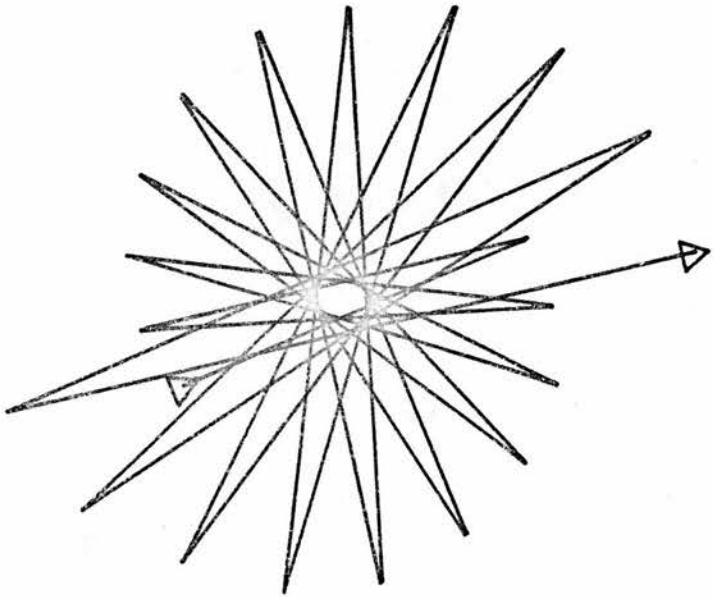


Figure 6.12. Sharply Pointed Spiral.

This spiral had an exterior angle of 170 degrees. The task here was not to write the procedure, which was given in the worksheet, but to

find appropriate arguments for it. She tried angles of 135 degrees, then 200 degrees and then 250 degrees. In the process she did not notice that, by passing through 180 degrees, she had inadvertently changed the sense of the spiral from clockwise to anti-clockwise and was now making the vertices blunter by increasing the angle. The author suggested that she try drawing spirals of 90, 180, 270 and 360 degrees which she did. Angles of 90 and 270 degrees produced 'square' spirals winding in opposite senses. While angles of 180 and 360 degrees were degenerate cases, producing an oscillating, expanding line and a lengthening line respectively.

When she returned to the problem of the sharply pointed spiral, she was able to suggest 160 degrees as a good value to try because it nearly sent the turtle back on itself. By small increments on the initial value of 160 degrees she ended up with a spiral, which she liked, with an angle of 170 degrees. This provides an instance where the reactive nature of the computer allowed her to explore angle properties and at the same time produce pleasant designs (in contrast to the rather unexciting 'house', described earlier). In this case she had spent a lot of time previously establishing how the recursive spiral procedure produced these effects, and in this session she was free to just run her spiral procedure with different angle argument values to observe the effects.

Both her exploration of the symmetry of the arrow and the sharpness of the spiral vertices were the result of her attempting the solution of some other larger problem, though it must be mentioned that both the work with 'trees' and with 'spirals' was suggested in the worksheets.

Linking Turtle and School Geometry

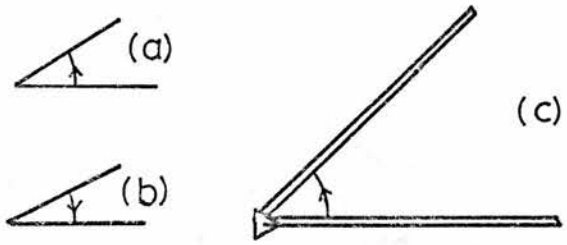
The following term, Jane was on teaching practice again. She taught two lessons in which she was observed giving the children practice in the use of ruler, compass and protractor to construct and measure triangles. In a programming session between these two lessons, her difficulties in the use of a protractor were observed. These have already been described in chapter 4.

These lessons and the intervening programming session will now be described because they show how Jane had failed to link her knowledge of angles as rotations (Turtle Geometry) to the use of the protractor (School Geometry). Only when the author had pointed out the link to her, was she able to make use of it in the way she taught the children.

The first of the observed lessons concerned the naming and construction of different classes of triangle, e.g. isosceles, using ruler and compass and the measurement of the angles of the constructed triangle. Most of the children appeared to be able to use the protractor competently, except for a minority whom Jane later reported were having difficulty.

At the start of the next programming session, Jane mentioned the difficulty she was still having teaching some of the children how to use the protractor. This difficulty was especially concerned with choice of the correct scale on the protractor. Part of this conversation was given in chapter 4. There it turned out that her difficulty in teaching this skill was due to her own misunderstanding of why the protractor had two scales. She had not linked the LEFT and RIGHT rotations of the turtle, with the clockwise and anti-clockwise scales on the protractor.

The author reminded her of the procedures LEFT and RIGHT and compared the rotations they produced with the protractor scales. She was encouraged to view angles, such as those in figures 6.13a,b as rotations by specifying, in discussion, sequences of programming commands which would draw them, see figure 6.13c.




```
FORWARD 200
BACKWARD 200
LEFT 45
FORWARD 200
BACKWARD 200
```

Figure 6.13. Angles as Rotations.

She did not actually program these commands, she merely wrote them out on paper. Establishing this link between the school work with protractors and turtle geometry proved beneficial as we shall see later in the description of a lesson observed subsequently. During the programming session she was asked whether she thought the earlier angle work had been helpful. She replied:-

> "Yes, well, it's just since I've been doing this that I realise, you know, I can sort of visualise an angle, you know if something is bigger than hundred and....ninety degrees. You know I can visualise how big its going to be. But before it was dreadful because I managed to get them all mixed up....I couldn't visualise them, no...unless I really, you know, thought about it hard for a long time."

Ten days later, Jane was observed in school giving two girls remedial help in measuring angles with protractors. This lesson showed that Jane had applied some of her programming knowledge. Jane

introduced the topic by asking the girls how they chose which protractor scale to use when measuring an angle. One replied, candidly:-

"I just use the first one which comes to me."

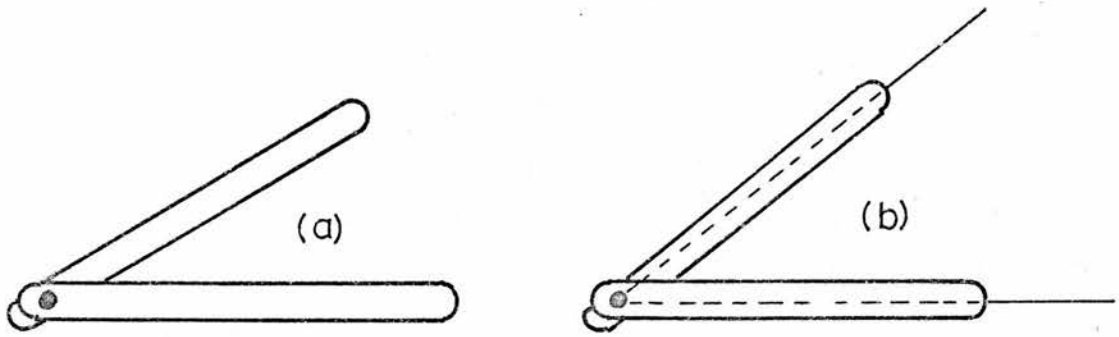


Figure 6.14. Teaching Aid.

Jane had constructed devices consisting of two cardboard arms joined with a paper fastener at one end, see figure 6.14a. She had made one for each girl and one for herself to demonstrate with. She showed the girls how the device could be used to illustrate angles as rotations. This was done by first closing the two arms and placing them over one arm of the angle to be measured, with the paper fastener on the vertex. Then one arm was opened (i.e. rotated) until it lay along the other angle arm, see figure 6.14b. Jane then tried to link this with the correct placement of the protractor and measuring from zero on the appropriate scale. Jane used the ambiguous terms 'left to right' and 'right to left' to describe clockwise and anti-clockwise rotations. Possibly these terms were derived from the turtle commands, LEFT and RIGHT. The girls attempted to match these terms to the inner and outer scales of the protractor. Unfortunately the girls only measured angles in triangles, and most of these triangles were orientated so that they had horizontal bases. This meant that the girls learnt rules of

thumb to measure the slope angles of the triangles correctly. So one girl still had trouble measuring the angle opposite the base and tried to place the protractor as in figure 6.15. It was measurement of this 'top' angle of a triangle which Jane had reported she found difficult, see chapter 4.

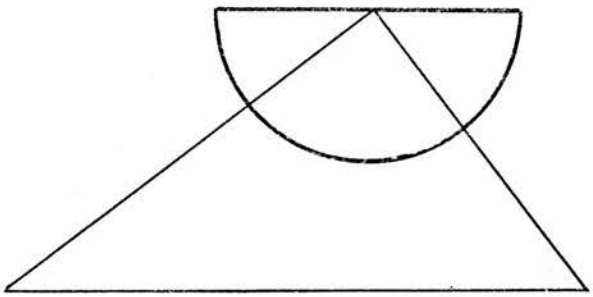


Figure 6.15. Top Angle of Triangle.

Later, Jane and Mary listened to the recording of this lesson together. The author pointed out the ambiguity of the terms 'left to right' and 'right to left' and the restricted range of examples of angles which had been used. Mary suggested that Jane should have given more emphasis to the idea of 'starting at zero' on the protractor scale, which might have prevented the children using the wrong scale. Jane seemed to have the idea that only one scale of the protractor was the correct one to use for a given triangle angle.

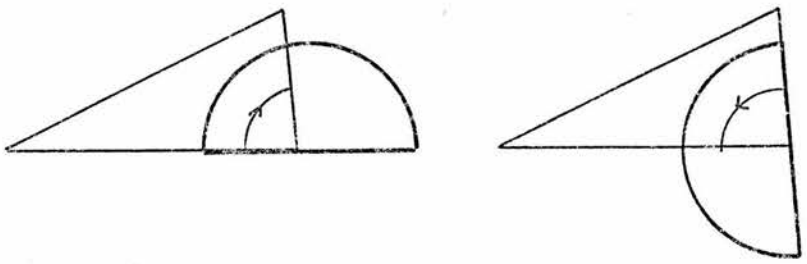


Figure 6.16. Using Either Protractor Scale.

When the author pointed out that either scale could be used for any angle (see figure 6.16) she admitted, with some hesitation, that each

way ought to give the same value for the angle.

The episode is revealing because it suggests that Jane had not linked the idea of angle as rotation, either clockwise or anti-clockwise, to angle measured with a protractor. This despite the extensive work she had carried out in Turtle Geometry. However when this link was discussed, making free reference to her acquired programming experience, she was able to make some use of it to design a piece of apparatus to help her pupils. Again this emphasises that the programming work should be accompanied by good teaching to help the students to make these links.

The next session also suggests that Jane had not linked together her school geometrical knowledge and Turtle Geometry. She was asked, as one question on a worksheet, to explain the relation between the angle rule for the triangle (that its angles sum to 180 degrees) and the fact that in a total circuit of a triangle the turtle turned exactly 360 degrees (the total turtle trip theorem). She was not able to do this. However when filling in a chart of the angle properties of various regular polygons (another worksheet question), she noticed that the total interior angles of regular polygons formed a pattern. As the number of sides of the polygons was increased by one, so the total of the interior angles increased by 180 degrees. She started her chart with the square, rather than with the equilateral triangle, and so did not notice that the triangle also fitted the same pattern. Although she had spotted the pattern, she was not able to explain it. She ventured:-

> "Has it got to do with...the angles on a straight line?...They are either one hundred and eighty or three sixty or..."

The problem was left unsolved for the moment. Jane then wrote a

procedure which could draw any regular polygon, given its exterior angle and number of sides, and also compute and print out the total of its interior angles.

By the next session Jane had still not been able to work out why the interior angle total increased by 180 degrees. Mary, who had also been working on this independently and discovered the reason, tried to explain to her but failed. Finally the author gave an explanation. Despite Jane's failure to solve the problem, something of mathematical benefit did occur. Jane spotted a pattern in the numbers in the chart and did try to find an explanation for this pattern.

Tessellating Polygons

Jane spent five sessions drawing tessellations of different regular polygons. This task would probably have produced better mathematical benefits if it had not involved programming. These sessions show the way that problems could be tackled at the wrong level of representation because of the availability of the computer. In this session she drew a ring of pentagons (see figure 6.17) and then tried to fill the centre of the ring with more pentagons. She found out that this could only be done if the pentagons overlapped, thus establishing that pentagons do not tessellate. In the succeeding four sessions she drew pictures of tessellating hexagons and octogons (which leave square spaces). However most of her, extensive, programming work was devoted to incrementally building up the desired picture rather than to an analysis of why certain polygons would tessellate and others would not. She maintained she enjoyed this work which demanded a great deal of angle drawing.

However from the point of view of understanding tessellation not much was achieved for all the effort. Indeed some simple work with cut out paper polygons might well have established the same findings with a fraction of the time and cost.

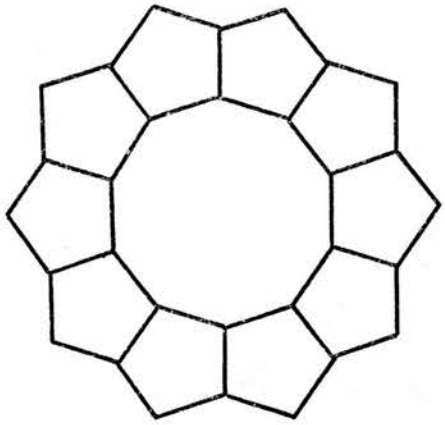


Figure 6.17. Ring of Pentagons.

The difficulty was that the programming problems of drawing pictures of tessellations was itself complex and intriguing. This diverted attention from the mathematical aspects of the situation. This may be seen as a failure of the author to make the best use of all Jane's enthusiasm. He did not direct her strongly enough to search out, for example, why hexagons tessellate and pentagons do not. Perhaps if Jane nad planned her pictures more thoroughly rather than incrementally building them, this point might have come out of her analysis.

6.3 ALGEBRA

In this section we describe Jane's work in various aspects of algebra. Some evidence is presented which supports the claim that programming, in particular with LOGO, can give insight into the key concepts of function and variable. It will be shown that the syntax

of LOGO provides an excellent illustration of 'function' and 'composition of functions'. An example is also given of the mathematics work Jane was tackling in the College of Education and how her programming work helped alleviate her difficulties, though not until the author had explicitly pointed out the link between the College work and the programming.

Parsing Mathematical Expressions

The programming worksheets which Jane used in her first term of programming introduced such names as 'state', 'generalise', 'name' and 'value' in the context of specific programming activities, though without direct reference to mathematics. That term Jane also learnt about the parsing rules of LOGO and about the use of parentheses to make the prefix commands of LOGO more easily readable. She compared her own evaluation of how a command would be interpreted with the result computed by the computer. For example, by typing the following command:-

PRINT SUM (PRODUCT 3 4)(DIFFERENCE 10 (SUM 3 2))

which would have the effect of computing and printing the result 17.

This prefix expression is equivalent to the more conventional infix expression given below:-

(3*4) + [10 - (3 + 2)]

Use of parentheses was not mandatory in LOGO commands, but it was recommended (especially because of Fiona's experience, see section 5.3 of Chapter 5). The quite explicit distinction between the prefix formulation of an arithmetic expression in LOGO and the more conventional infix notation is a valuable asset because it forces the issues of the binding, scope and parsing of arithmetic operations

into the open. It was for this reason that the infix arithmetic operations were removed from the final implementation of LOGO.

The intention of these exercises was to illustrate the idea of 'parsing' a mathematical expression, by asking Jane to translate backwards and forwards between two notational systems. Further examples of this kind were undertaken by Jane when she returned to programming after her first teaching practice in the summer, 1976. In the questionnaire (2) administered that term, Jane mentioned this work in her answer to the question about the value of the programming work in her future teaching. She mentioned the work with angles (see section 6.2) and also:-

> "...use of brackets in some procedures to make an inner procedure to make the answer easier to find--I find that using the computer to do them much more enlightening than just being told about it."

Her use of the word 'inner' suggests that she has grasped one of the essential uses of parentheses.

Variables and Functions

In her study of variables, Jane wrote a number of procedures taking arguments. Jane extended a given rectangle procedure to compute and print out its area and perimeter. She had some difficulty in forming syntactically correct commands to compute the area and perimeter, although she was familiar with the algorithm for their calculation. The main difficulty was in her use of the name of the variable instead of its value. Eventually, with some help from the author she sorted this out. She chose to use parentheses in the commands which computed area and perimeter.

After further practice in using argument names and values, Jane wrote a number of procedures which acted like functions. These took a single argument and computed a single result, which if necessary could become the value of the argument to another procedure. One such function-like procedure was to square numbers (i.e. 3->9, 5->25). In LOGO such a procedure, named SQUARE would be used as follows:-

```
W: PRINT SQUARE 5
 25
```

Here the prefix notation of LOGO and Jane's experience in parsing commands made it easy for her to understand both flow of control and result passing in the composition of SQUARE procedures below:-

```
PRINT SQUARE SQUARE SQUARE SQUARE 2
```

though she was a little surprised at the number printed, 65536.

Author: "What did you work it out to be?"

Jane: "I thought maybe about thirty-two."

Author: "Yes right that...yes,yes...Do you know why it isn't thirty-two? Why?"

Jane: "Because you keep, you keep having to square the number."

Author: "Right."

Jane: "And I kept multiplying up by two."

The author pointed out that this was a common mistake, by explaining how he had jumped to the same conclusion when first encountering this exercise! Jane continued:-

"I find I always make mistakes with squares because, huh, we had to do the slide rule in one of our maths things and when it said square, I just multiplied by two, forgetting that you are

actually having to multiply by...by itself...I always...the minute I see square, I always think its because, I think of the little two at the side. Nine squared, you can write it as nine with a little two. I always think of multiplying by, by two instead it means multiplied by itself."

This incident had the value of underlining the difference between squaring and doubling because of the disparity between Jane's hypothesis of what the computer would print and what actually was printed. They are easily confused because of their similarity of notation:-

5^2 and 5×2

LOGO's rather long-winded formulation of the procedure for squaring a number is quite different from the procedure which will double a number. Compare the two procedures below, which have been given the appropriate names:-

```
DEFINE "SQUARE "NUMBER
10 RESULT PRODUCT VALUE "NUMBER VALUE "NUMBER
END

DEFINE "DOUBLE "NUMBER
10 RESULT PRODUCT VALUE "NUMBER 2
END
```

Another value of programming in the teaching of functions is that, in a prefix language like LOGO, it is quite straightforward and natural to compose functions, as in the multiple use of SQUARE, above. Also the passing of results in the command corresponds with that function notation convention which interprets fg(x) as applying function g to x, and then applying function f to the result. This point, misunderstood by both Jane and Mary, will be returned to later.

Equation Solving

After the summer vacation, Jane was again on teaching practice and this time also attending programming sessions. At her first session that term she explained a difficulty which had arisen on her first morning of teaching practice. She had been asked to help children do problems in equation extraction and solution. Her difficulties with this were described in chapter 4. Briefly she found it difficult to extract a single variable equation from a problem stated in English and difficult to solve such an equation once it was extracted.

In attempting to help her, the author made use of her previous programming experience of names, values and parentheses. He pointed out that solving a story problem consisted of solving two sub-problems. The first was to transform the given English story into an equation. The second was to successively transform the equation until it reached a form which could be solved by inspection. It was explained that the 'letters' used in the equations in the book, she had brought from the school, were 'names', similar to argument names in programming. It was also pointed out that the process of solving such an equation was that of finding the appropriate value to associate with that name. Jane was shown how parentheses could be used to mark successive stages in the extraction of an equation from the text and then tried the following problem herself.

"I am thinking of a number. When 5 is subtracted from six times the number, the answer is 37. What is the number?"

With a little prompting from the author, Jane wrote out the following sequence of expressions in which the variable was gradually

imbedded more deeply by means of parentheses. The author suggested she use the name 'number' for the variable. The following diagram shows what Jane wrote together with the appropriate phrase from the problem text.

EQUATION EXTRACTION	PROBLEM TEXT
number	I am thinking of a number
(number * 6)	six times the number
((number * 6) - 5)	when five is subtracted from
((number * 6) - 5) = 37	the answer is 37

Solving the equation was shown to be a reverse process of evaluating the parentheses:-

((number * 6) -5) = 37

(number * 6) = 42

(7 * 6) = 42

Jane was asked to express the equation more conventionally, and correctly wrote:

6x - 5 = 37

This work used a variety of terms derived from programming and was conducted 'off the cuff' in response to Jane's enquiry.

Symmetry Transformations

Jane's work with tessellations was an example where the programming task was intriguing but mathematically unproductive. In contrast, the following work on transformations was ideally suited to exploration through (LOGO) programming. The distinction between 'state' and 'transformation' and the idea of 'composing' transformations emerged clearly. The programming details were reduced by providing the students with new primitives for this

domain. Their task was to run combinations of these primitives and observe their effect.

After teaching practice Jane was again attending lectures in the College of Education. Both Jane and Mary complained that they were finding some of their mathematics course hard to follow. They had both chosen to take mathematics as one of their principal areas of study because they knew that they needed extra help. Others taking the same course had been more successful in mathematics and it appeared that some topics chosen in the lectures, while suitable for the majority, were causing Mary and Jane some difficulty. In particular they complained about some work based on groups, including the group of symmetry transformations of the rectangle. Jane felt worried both about the overall objectives of this work and about how she was to do the questions on a worksheet supplied by the College.

> "Well I'm just not sure exactly what we are supposed to be doing. Why, I mean...what is this that we are trying to do? I don't see it at all...and I'm not sure how to do it anyway. I mean I have done bits of it, but I don't see why I am doing it. I think you are following a formula but I don't see why I am doing it."

Her comment 'following a formula but I don't see why' is like many of her remarks, given in chapter 4, which reflect her knowledge of mathematics as rules without understanding the reasons for those rules. Two weeks later the author had prepared a worksheet and written procedures based very closely on the examples and notation in the worksheet from the College. The four symmetry transformations of the rectangle were modelled in four procedures which transformed and drew a rectangle on the display screen. One corner of the rectangle

was marked with a spot so that the effect of a transformation could be detected. The transformations and their names and effects are given in figure 6.18.

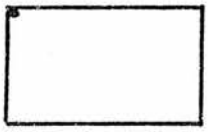
I Identity transformation

X Turn over about horizontal axis of symmetry

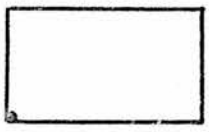
Y Turn over about vertical axis of symmetry

Z Rotate in own plane through 180 degrees about centre

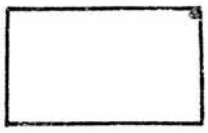
DRAW RECTANGLE



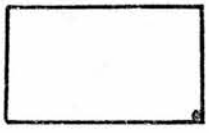
DRAW I RECTANGLE



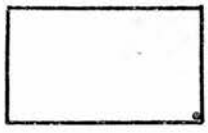
DRAW X RECTANGLE



DRAW Y RECTANGLE



DRAW Z RECTANGLE



DRAW X Y RECTANGLE

DRAW X Y Z I RECTANGLE

Figure 6.18. Symmetry Transformations.

The author wrote these procedures because they were beyond Jane's ability to construct and because it was felt that writing procedures to illustrate a transformation by drawing was not necessarily the best way for Jane to understand the properties of the transformation. The procedures were written in LOGO, in such a way as to allow the composition of the transformations, e.g. a reflection followed by a rotation. The worksheet from the College of Education invited the student to fill in a group table for this set

of transformations together with the binary operation 'followed by', see figure 6.19. They were also given a graph whose four nodes corresponded to four orientations of the rectangle and were asked to label the arcs between the nodes with appropriate transformations, see figure 6.20.

followed by	I	X	Y	Z
I	I	X	Y	Z
X	X	I	Z	Y
Y	Y	Z	I	X
Z	Z	Y	X	I

Figure 6.19. Group Table.

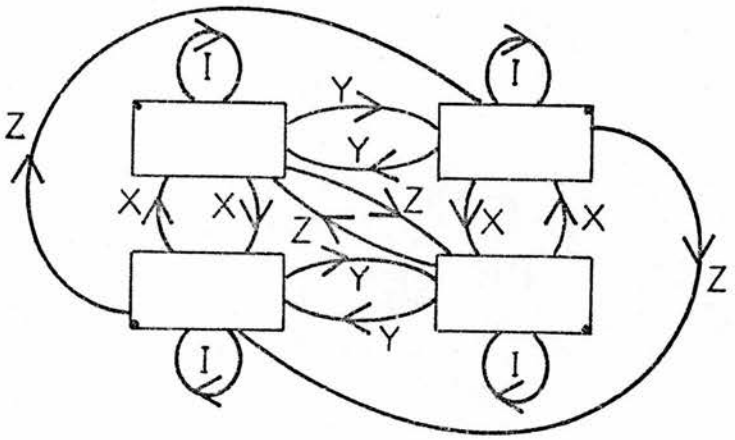


Figure 6.20. Graph of Transformations.

One of Jane's original difficulties had been that she had not clearly distinguished between a transformation of the rectangle and the state of the rectangle before and after a transformation was applied. She had labelled the nodes as well as the arcs with the

names of transformations. Another of her difficulties concerned composition of transformations. The College of Education worksheet explained that in its notation ZX meant first do transformation X and then do Z to the result. What Jane had not fully grasped was that Z was to be applied to the rectangle's state produced by first doing X. She thought that one chose a state, did X, then chose the initial state again and did Z. She also explained about the difficulty she had earlier, trying to explain to her College lecturer what she did not understand:-

> "Well, you see the bit that I tried to explain to Mr. B. [in the College of Education] that I didn't understand, but he didn't understand what I was trying to explain...was that when you do one operation followed by another, how isn't it that you just get the, the last operation as your answer? For example if you do X followed by Y, why isn't it that its just Y that's the answer?"

In her system every composition would be equivalent to the transformation applied last of all. That is:

$$ZX = Z$$

$$XYZ = X$$

This latter difficulty was especially apparent when she had to make up entries for the group table, see figure 6.19. Jane reported her difficulty as follows, in relation to a similar piece of College work on the symmetries of an equilateral triangle:-

> "...I could see when we had a triangle and were twisting it around in different ways. But when you were making up the tables, that's when I really get lost. I don't see how, you know, he [the College lecturer] gets all these letters...I

really get lost with that."

She went on to explain that she understood how each transformation would allow one to pick up the shape and replace it in an identically shaped frame. She also realised that different shapes would have different sets of such transformations, but she added:-

"...but its when you put them in a table, that's when I find it hard...you know, one following another."

During this session and the next Jane worked on the worksheet which provided a visual illustration, using the procedures, for the work set by the College.

The programming work was valuable because it distinguished between the state of the rectangle (as seen on the screen) and the transformation (typed in as part of a command). The prefix syntax of LOGO exactly matched the notation adopted by the College. Jane then knew about result passing (see the SQUARE example earlier) and the author was able to point out the similarity.

However Jane had not perceived this similarity for herself. In this way it was like the protractor/angle example seen in section 6.2. Jane had covered all the programming prerequisites but had not made the link between the College work and the programming work. Once the link was pointed out, however, she was able to understand it.

In the course of running the procedures written by the author, Jane spotted a bug. This was that transformation Y did not leave the rectangle in the correct state. Jane pointed out that Y was having the wrong effect, and the author fixed the bug. This suggests that although Jane was only running author-defined procedures, she was observing their effects carefully and matching these effects to the

worksheet text.

In the second session the difficulty about composition of transformations was attended to. Jane tried out a variety of commands which involved two transformations to see what single transformation could have produced the same effect e.g.:-

DRAW X X RECTANGLE

This had the same effect as

DRAW I RECTANGLE

The author explained the composition of transformations by asking Jane to consider the overall effect of two transformations, 'painting the rectangle blue' and 'turning the rectangle upside down'. After the discussion Jane explained that she now knew how composition worked.

"Oh I see it's the two together"

"So its the result of both of them together"

She was then able to fill in the group table, see figure 6.19, and to solve questions involving finding a single transformation equivalent to a composition of two or more transformations. For example, she found that there was no transformation P which satisfied the equation below:-

PX = P

However she still had labelled her graph incorrectly by marking both arcs and nodes with the transformation names I, X, Y and Z. Her difficulty had been made worse because the arcs for the identity transformations had been omitted from the diagram given to her. The difficulty was further aggravated because there was a preferred initial state for the rectangle, produced by the following command:-

DRAW RECTANGLE

This command specified no transformation and merely drew the rectangle. Jane labelled the other rectangle states on the graph with the names of the transformations, which would produce them from the initial state. What she had not fully grasped was that the particular initial state of the rectangle was immaterial to the relations between the transformations that held for all possible rectangle states. That is the statement:

$$XY = Z$$

was true whatever the initial state of the rectangle. The author tried to explain this point but Jane took little active part in the conversation, merely saying 'uhu' at intervals, so the author was unable to gauge how far she had understood. In this respect, Jane behaved quite differently from Mary, who would not let the author explain 'at' her but would take an active part in the conversation by demanding further information or by explaining back to the author what he had just said. At the end of his explanation Jane reported that:-

"I feel a bit more confident with these ones"

But she said she was still a little worried about dealing with other shapes. She mentioned the 'Isle of Man' symbol in particlar.

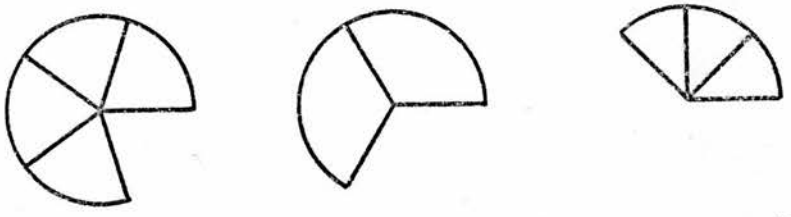
6.4 NUMBER

In chapter 4 we saw that Jane was worried because she knew a number of mathematical rules but did not why these rules worked. A typical example was the rule about dividing fractions. This section describes how Jane used procedures, provided for her, which gave a visual illustration for such rules. An example is also given of a project with fractions which was abandoned and which shows how it is

possible for the programming aspects of a problem to overwhelm the mathematical aspects of that problem. Part of the success of using pre-written procedures was that they allowed Jane to explore mathematics difficulties at the right level of representation. In the abandoned fraction project she spent her time worrying about drawing fraction diagrams rather than worrying about fractions.

Drawing Fraction Diagrams

Jane attempted two worksheets concerned with fractions. The first worksheet attempted to show how fractions were a new type of number built on the natural numbers. It also attempted to show how fractions could be considered as a 'double-operation' of division and combination. This latter was to be illustrated by the student defining a general procedure, named FRACTION, to draw fractional parts of a disc, see figure 6.21.



FRACTION [4 5] FRACTION [2 3] FRACTION [3 8]

Figure 6.21. Parts of a Disc.

This procedure took a single argument whose value was a list of two elements representing the numerator and denominator of the fraction respectively. Students were intended to break down the problem of drawing such a diagram into sub-problems. One sub-problem was to draw of a single 'slice' of the disc (illustrating the division aspect of a fraction) and the other was to combine a number of these slices together (illustrating the multiplicative aspect of a

fraction).

When Jane attempted the problem she sub-divided the problem in a quite different way. She tried to draw the 'spokes' of the chart separately from the arc of the circle, as in figure 6.22.

Figure 6.22. Problem Decomposition.

Although this was not what the author expected, or wanted, he decided to let Jane pursue it. The procedure she started to define took the radius of the circle and angle between the spokes as two arguments. These arguments were expressed in units of length and in degrees and were not the dimensionless quantities suggested in the worksheet. By the end of the session, Jane had still not fully debugged her procedure which would work only for a particular angle argument value. This was the last session of the term before the Christmas vacation. About a month later, Jane returned to the problem and went some way further in debugging the procedure. This session ended suddenly with a power cut, the whole building was plunged into darkness and the system crashed.

The next session lasted only fifty minutes before there was another system failure. In that time Jane got little further with the problem than in the last session. In the end the worksheet was abandoned and the next session Jane went on to the work on transformations of the rectangle, described in the last section.

Despite the unfortunate sequence of crashes, this particular project was unsuccessful because Jane had planned a different programming solution than that expected by the author. She had also

become immersed in the problems of producing one of the pictures given as an example in the worksheet, as in the tessellation project described in section 6.2. Both of these factors diverted her attention away from the proposed objective of understanding fractions.

Putting Meaning into Fraction Rules

Much later Jane worked on fractions again, this time running procedures written by the author. These procedures were intended to illuminate a meaning for multiplication and division of fractions which Jane had explained she did not understand:-

Jane: "Why is it, when you div...why is it when you divide"

Author: "You turn it upside down and multiply."

Jane: "I don't understand why you do that because...we did the division...we did it last term at [College] and the tutor, I had, you know, we all asked why and he said it was obvious why, and never really explained."

Author: "It's not the least bit obvious."

At the time the author made some general comments about addition and subtraction of fractions being easier to illustrate than multiplication and division. It seems that the author's own attempt to explain the rule for dividing fractions, in a seminar in the first term, had not been remembered. Eventually a worksheet and a set of procedures were written by the author which addressed this issue and which provided an illustration for multiplication and division of fractions.

Jane was given a procedure to run which drew a simple house on the display screen. The procedure then requested her to type in two

numbers representing the numerator and denominator of a fraction. This fraction was then used by the procedure to draw a second house of similar shape but changed in size according to the value of the fraction. Thus an entry of '3 4' would produce a second house three-quarters the size of the first, while a subsequent entry of '2 1' would produce a third house twice as large as the second, see figure 6.23.

Figure 6.23. Multiplication of Fractions.

This process could be continued indefinitely, so long as the student did not enter a fraction such that the resultant house was too large for the screen.

The essential feature of this procedure was to provide a ratio or exchange interpretation of fractions (Kieren, 1975) which could be used to illustrate multiplication by composition of exchanges and division as the inverse of multiplication.

Jane's activity consisted of running this procedure in response to questions posed in a worksheet and of discussing the meaning of the questions and the visual effects with the author. One reason for providing such a procedure rather than suggesting Jane wrote one herself was the long gap since she had last programmed. A second reason was the failure of her last fraction project where programming considerations had overwhelmed the mathematics.

After finding out how to run the procedure and the kinds of effect it could produce, Jane attempted to answer questions on the worksheet:

> Find out what fractions leave the size of the house unchanged (the identity).
>
> Find out what fractions have the same effect as 3 4 ie 3/4.
>
> Find out what fraction exchange undoes the effect of 3 4 (the inverse).

Jane already knew something about fractions and had ideas on what the answers to some of these questions would be. The session did not consist of her 'discovering' entirely new fraction concepts but of her confirming, clarifying and putting meaning into the various fraction 'jingles' which she knew ('turn upside down and multiply').

She saw that fractions such as 1/1, 6/6 did not change the size of the house and described how an infinite set of such fractions could be generated. She had some trouble, initially, in deciding on an equivalent for 3/4. Part of the trouble seemed to lie in her lack of understanding of what was required. She suggested that three quarters multiplied by itself might be equivalent but then rejected it because it produced a smaller house. She then discovered the equivalence:

$$3/4 * 1/1 = 3/4$$

This was like earlier work with symmetry transformations where the task had been to find a single transformation equivalent to a given pair of transformations. Now Jane appeared to be looking for a pair of fractions, to be multiplied together, which were equivalent to the single given fraction, 3/4. At this stage the author explained what was required, using hand drawn schematic diagrams based on the

display pictures, see figure 6.24a.

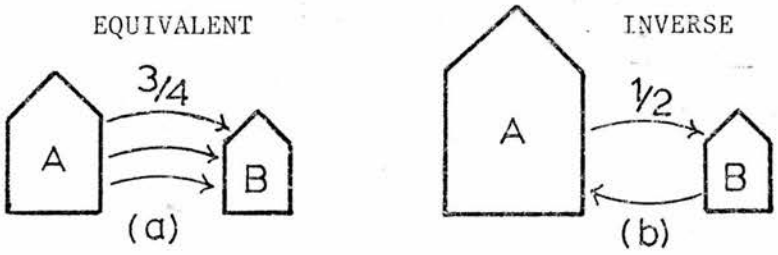


Figure 6.24. Equivalent and Inverse Fractions.

Using the diagram he showed how one 'path' from house A to house B was via the exchange 3/4. He asked her what other 'paths' could be found which linked the two houses. She was then left to solve the problem, which she did having tried 6/8 and 9/12 using the procedure. Again she gave a rule for generating an infinite set of equivalents for 3/4, and related this rule to that for finding equivalents for 1/1, which she said was "just the same idea".

The author asked her whether she knew the phrase 'family of equivalent fractions' or 'set of equivalent fractions'. She replied:-

Jane: "Yes, we have done that. I know what it means now, because I was never really clear exactly before...exactly what that meant. But it's just the same fraction multiplied by the...the same as a fraction multiplied by the same number again and again."

Author: "Em, yes, you mean above and below."

Jane: "That's right."

Her explanation of equivalence is a little ambiguous, and is expressed as a rule for generating equivalent fractions rather than in terms of the visual exchange interpretation. Nevertheless she now

seemed to understand what equivalence meant.

When she was looking for the inverse of 3/4, the author again used the schematic diagrams to suggest that the inverse of a fraction was the 'path' back from the second house to the first, see figure 6.24b. This time she suggested that 3/4 might be its own inverse but rejected it without use of the procedure because she realised that

$$3/4 * 3/4 < 3/4$$

She then said:-

"So would you have to change the fraction round"

The author was neutral about this suggestion and left her to work with the procedure. She tried the sequences schematically indicated in figures 6.25a,b.

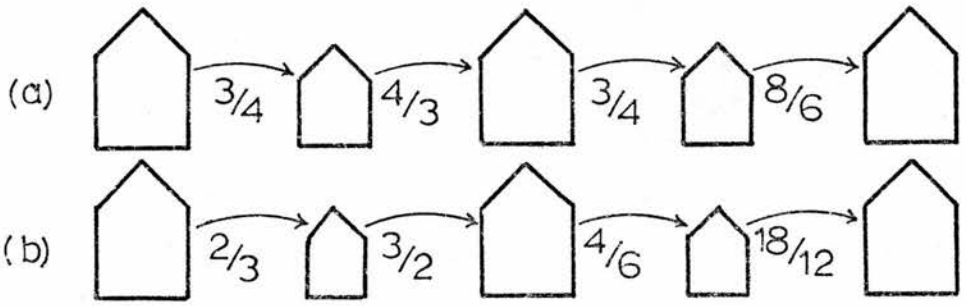


Figure 6.25. Fraction Sequences.

When the author returned she explained that:-

"You just have to turn the fraction round."

She also gave some examples. But she also added, unprompted that "any equivalent fraction" could be used.

Both her exploration with the procedure and her answer suggest that she had formed a clear idea of equivalent fractions and was able to see, for example, that 18/12 was the inverse 3/4. When asked why she had tried 4/3 as the inverse of 3/4 she replied:-

"If you divide a fraction by itself turned round you get back to

one."

This was incorrect since she should of said 'multiply' for 'divide'. However it was passed over at the time, since Jane did link together ideas of 'division', 'inverse' and 'getting back to the beginning'.

The last question on the worksheet was for her to find a single fraction equivalent to the pair 3/4 * 1/2. Again the question was described schematically as in figure 6.24. She knew that the answer should be 3/8 and confirmed it using the procedure. She did this by showing that 8/3 was the inverse of 3/4 * 1/2. That is, it exactly undid the combined effect of 3/4 and 1/2.

This session ended at this point. Jane said that she had enjoyed the session. Her demeanour throughout had been rather different from previous sessions. This time she took a much more active part in conversations with the author, proferring explanations and enlarging on what he had said. In earlier sessions she had tended to listen politely to an explanation but not try to re-explain back to the author.

The next session built on what had been already achieved to illustrate a meaning for fraction division using the same schematic diagrams and by running the same procedure. In the course of answering the questions on the worksheet, Jane was able to evaluate correctly the expressions :-

$$(1/2)^2 \text{ and } (1/2)^3$$

She mentioned that she no longer muddled up squaring and doubling as we saw earlier. The worksheet used the term 'commutative' which she could not remember the meaning of, except that it had something to do with addition. When she was reminded of the meaning she clearly

understood that fraction multiplication was commutative.

To illustrate fraction division, the worksheet suggested that a division should be restated as a multiplication, as below:-

$$8/2 = X \Longrightarrow 2 * X = 8$$

$$(2/3) / (1/4) = X \Longrightarrow (1/4) * X = (2/3)$$

In each case the division is solved if the variable in the multiplication can be found. Later she mentioned that she had to check such restatements for the case of fractions by matching them against an example using natural numbers. A similar strategy was also observed in her mathematics test answer to a formula rearrangement, see chapter 4. The multiplications derived from these rearrangements of the problem were to be illustrated using the same procedure as in the last session, see figure 6.26.

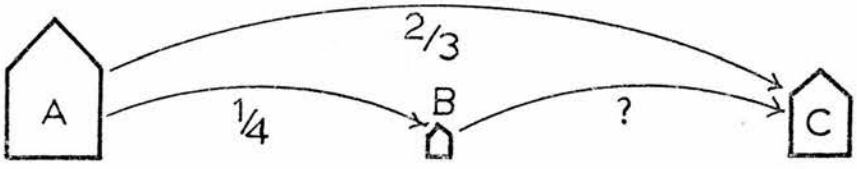
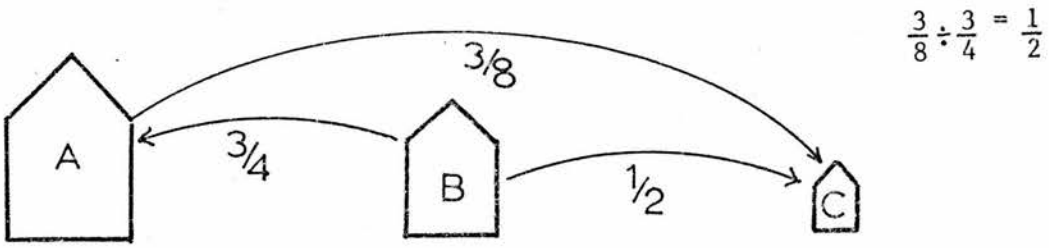


Figure 6.26. Division of Fractions.

Jane spent some time trying to find the value of the 'path' marked with a '?' in the diagram, using the procedure. Finally the author re-explained the problem in terms of the schematic diagram. That is to find the missing fraction between house B and house C, one could take the alternative but equivalent path from B to A (using the inverse) and then from A to C. Suddenly she understood:-

> "Oh I see, so its just four over one multiplied by two over three."

What she had not appreciated earlier was that the exchange B->C was equivalent to the exchange B->A->C, see figure 6.27.



$\frac{3}{8} \div \frac{3}{4} = \frac{1}{2}$

Figure 6.27. Equivalent Paths.

Jane then mentioned the problem of teaching children fractions and worried that this presentation of fraction operations was not applicable in the classroom.

"The thing is could you explain that to children?"

"How often...I mean, I have never seen a teacher do it and its a case of there's a rule and the children have to follow it because I don't think they ever do drawings where they have to change the shape."

Jane's comments illustrate a difficulty of teaching mathematics to student teachers. She was worried that she could not apply the approach used here, in her own classroom. Her concern was not whether she understood this fraction concept, but whether she had a technique available to teach the children.

The author suggested ways in which the presentation might be adapted for the classroom and reiterated his belief that one had to try to teach for understanding. Otherwise one would only produce more students, like Jane, who knew rules but did not know reasons for those rules.

A little while later she expanded on the same theme, in a way

which suggests that she had benefited from the fraction work:-

> "I know myself that if I had to...probably if I hadn't done this and some child had asked me...I wouldn't of...aha, I would say it's just a rule. And I suppose a lot of teachers maybe just have to say that...it's just a rule."

Coordinates and Vectors

This section describes Jane's work with coordinates. It shows how work with coordinates, taking negative integer values, was incorporated naturally into her programming work because it solved the problem of accurately positioning a picture on the screen. In this case mathematics was seen as useful in solving a problem, as it had been in the case of drawing an 'arrow' (see section 6.2).

During the summer vacation, Jane was given a worksheet on coordinates to help her position pictures on the display screen more easily. She found the worksheet hard to follow at first since she had very little prior knowledge of this topic. The author spent some time explaining coordinates both in Turtle Geometry and in other contexts, e.g. map reading. Jane was still anxious and decided to postpone work on this topic until the next session. She found new terms such as 'axis' confusing.

The next session, Jane successfully defined a procedure of two arguments, named GRID. This procedure centred the turtle on the display screen would move it anywhere within the top right-hand quadrant of the screen by using the two arguments as x and y coordinates, see figure 6.28.

```
DEFINE "GRID "ALONG "UP
10 CENTRE
20 LIFT
30 FORWARD VALUE "ALONG
40 LEFT 90
50 FORWARD VALUE "UP
60 DROP
END
```

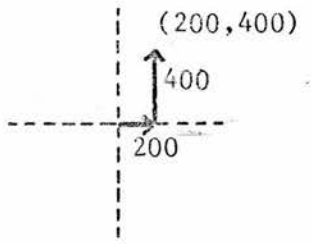


Figure 6.28. Coordinate Plotting Procedure.

Although Jane understood that this procedure could be used to position the turtle anywhere in the shaded portion of the screen, she suggested that different procedures would have to be defined to move it to the unshaded parts. She did not realise that, merely by giving her procedure negative argument values, the turtle could be moved anywhere on the screen. The author then introduced the idea of negative integers and suggested that she find out the effect of giving negative input to the drawing procedures FORWARD, BACKWARD, LEFT and RIGHT. When he returned, she reported:-

"Well a minus seems to make it go in the opposite direction."

But she still did not see the relevance of this to the coordinate problem and asked:-

"So why do you use minus numbers then? Why can't you just say backward ten [instead of forward minus ten]?"

The author did not answer the question but left her to think about it for herself. She then tried out her coordinate procedure with a variety of negative and positive integer arguments and found out that she could in fact position the turtle anywhere on the screen.

In discussion with the author she was able to explain the relation between the sign of the argument values for her procedure and the quadrant into which the turtle would be sent, as in figure 6.29.

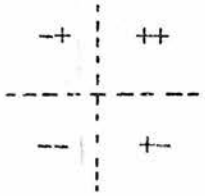


Figure 6.29. Signs of Coordinates.

She also explained that when working on this problem at home, before the session, she had decided to place the origin of her coordinate system in the bottom left-hand corner of the screen, rather than in the centre. This would have obviated the need for negative integers. This was a valid solution to the problem, but the author wanted her to see how negative integers could be useful. This proved to be the case because she used her procedure GRID in subsequent sessions, with negative arguments, to position the turtle (for example when drawing spirals).

In the Spring term 1977, Jane had to teach single lessons on specific topics to small groups of children as part of her College of Education mathematics course. For one of these lessons she had been directed to teach vectors and was rather worried about it. She described a game on a grid in which children were to use two element vectors to make moves. The author discussed this with her and related it back to the coordinate work she had done before, including the work with negative integers. It was suggested that the children could write 'procedures' consisting of sequences of vectors which could then be plotted to produce outline shapes as in figure 6.30. The similarity of this to LOGO drawing was also brought out. It was also suggested that the turtle state of [position, heading and pen inclination] could be considered, by Jane, as a three dimensional vector.

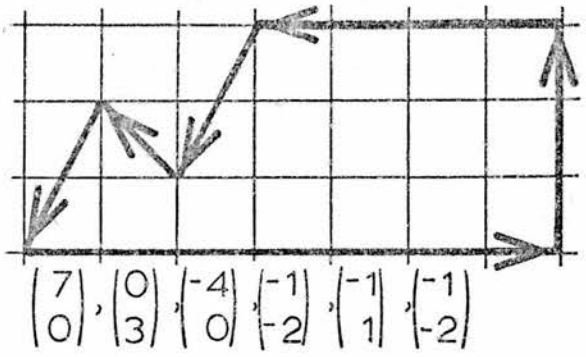


Figure 6.30. Sequence of Vectors.

However the discussion rather muddled up the concept of coordinate with that of vector. The distinction between vectors and coordinates was further confused by the discussion of integers which were illustrated as directed numbers. These were specifically explained as one dimensional vectors showing '+' and '-' as signs indicating direction (Williams and Shuard, 1970). (Controversy surrounds the teaching of integers, with some advocating an axiomatic approach and others favouring the more visually illustrative directed number approach. For a discussion of this issue see Gardner (1977) or Leddy (1977)).

Some work was undertaken with vectors by considering turtle movements as relative vectors. This was a mistake since turtle movements did not behave like vectors, e.g. addition of movements was not commutative. This meant that Jane was working with an incorrect set of primitives. The failure of this work was similar to that of the fraction pie-chart drawings, considered earlier. In each case the task of the student was to draw pictures representing the structure in question. But commands to draw a representation of a structure (e.g. a vector) are not the same as commands to manipulate that structure. In the same way, commands to manipulate symbols representing fractions may reveal little about fractions. Jane was

asked to draw the pair of vectors AB and BC, in figure 6.31, using LOGO.

```
FORWARD 300
LEFT 90
FORWARD 400
```

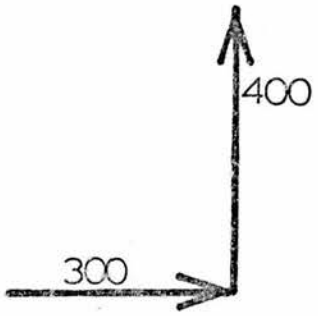


Figure 6.31. Drawing Vectors.

She was then asked to find their sum by seeking for the single command which would produce a line corresponding to AC. She soon discovered that the magnitude of AC was not equal to the sum of the individual magnitudes of AB and BC. She did not automatically calculate magnitude using Pythagoras' theorem but tried a convenient first order approximation that slope angle was 45 degrees and that the magnitude was given by $|AC| = |AB| + |BC|$. By a process of successive refinement she found the best answer on her sixth attempt, see figure 6.32.

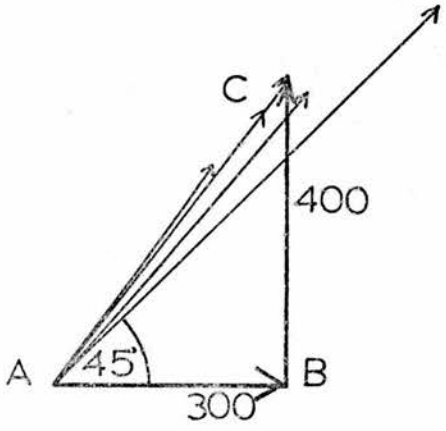


Figure 6.32. Finding The Sum by Successive Refinement.

Jane spent much time on another question which asked her to show that addition of vectors was commutative, using LOGO drawing. She

tried to do this by driving the turtle back from C to A, using the same figure as the last question, above. She was plotting the opposite vectors CB and BA. She failed to draw these two vectors because she could not get the heading of the turtle correct at point C, see figure 6.31, to draw the first vector CB. Instead she produced figure 6.33. In this figure the numbers 1,2 and 3 indicate three attempts to draw the vector.

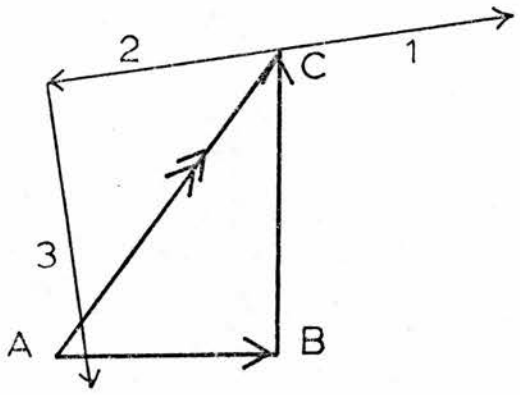


Figure 6.33. Three Attempts to Draw a Vector.

This work provides an example of the inappropriate use of Turtle Geometry primitives where vector primitives should have been provided.

Modelling Integers

The integer work shows how the standard drawing primitives FORWARD and BACKWARD were successfully used to model integers and integer operations via the idea of directed numbers. It also shows how Jane was able to solve problems with this modelling system and also how she was able to extend the scope of schematic diagrams based on the system.

The model for a positive integer was a call on the procedure FORWARD, and for a negative integer a call on the procedure BACKWARD. Each new sequence of calls started at the centre of the graph-plotter

paper (graph-plotters were used for this work rather than the display, because it enabled Jane and Mary to sit next to each other, each with a graph-plotter). The model for addition of such integers was to follow one movement by another, starting the second movement from where the first finished, see figure 6.34. In the figure the movements are arrowed and distinguished vertically. When Jane (and Mary) drew them they were plain lines, sometimes passing over each other, if the paper was not moved vertically.

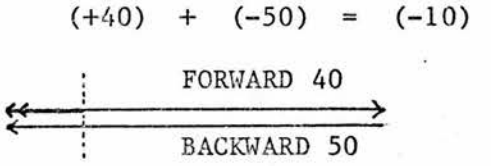


Figure 6.34. Model for Integers.

At first Jane misunderstood the model. She thought that, in the model, the sum of the movements was that movement which would return the turtle pen to the starting point see figure 6.35. She did not see that it had to be that single movement which would have the same overall effect as the component movements taken together.

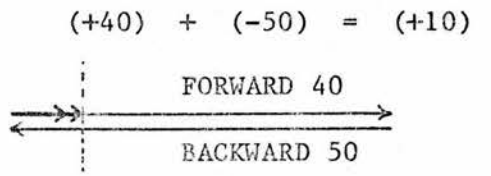


Figure 6.35. Incorrect Sum of Two Integers.

The author did not realise initially that Jane misunderstood the model. Early questions of hers should have alerted him:-

Jane: "Do you still call it minus even though it's not really minus?"

Author: "Now what do you mean, it's not really minus?"

Jane: "Well it's not really addition and subtraction but well that's...that answer there, that vector."

Her problem here appeared to be the common confusion between addition and subtraction as operations and plus and minus signed integers.

Jane and Mary were sitting side by side working individually on the same problems on two graph-plotters. Jane looked over to Mary and saw that she was getting different answers to the worksheet questions. Jane then corrected Mary and the following conversation brought out the nature of Jane's mistake:-

Jane: "That should be plus fifty."

Mary: "No, whatever is on your left-hand-side is minus, isn't it."

Jane: "But you are having to go forward to get back to the centre again."

Mary: "So, if you want to go back to the centre, you have to tell it plus fifty."

Jane: "So it's not minus fifty."

Mary: "It is. I mean, how, I mean, in which direction has your plotter moved? That is what you are trying to find out."

Mary had concisely stated what needed to be done to find the sum: that is, see how much and in what direction the plotter pen had moved as a result of the given sequence of commands. This was contrasted with Jane's view that the sum of the movements was that movement needed to 'undo' the effect of all the commands. With some further explanation from the author, Jane understood Mary's point.

The next exercise asked the students to write 'banking' procedures named DEPOSIT and WITHDRAW which were intended to model

'movements' of money into (plus) and out of (minus) a bank account. The author had intended the students to increment and decrement a variable. But Jane solved the problem by using the drawing model above. Her procedure for depositing money moved the plotter pen forward, to the right of the paper, and her withdrawal procedure moved the plotter pen backward, to the left of the paper. The balance was shown by the pen's current position. To the left of the centre the account was in debit and to the right it was in credit. Jane needed a little help in debugging her procedures because of programming errors. The finished procedures are shown below and part of the sequence of Jane's use of them is given in figure 6.36.

```
DEFINE "DEPOSIT "ACCOUNT
10 FORWARD VALUE "ACCOUNT
20 PRINT [THE BALANCE IS]
30 PRINT XCOR
END

DEFINE "WITHDRAW "MONEY
10 BACKWARD VALUE "MONEY
20 PRINT [THE BALANCE IS]
30 PRINT XCOR
END

W: DEPOSIT 20
 [THE BALANCE IS]
 100
W: DEPOSIT 30
 [The BALANCE IS]
 130
W: WITHDRAW 70
 [THE BALANCE IS]
 60
W: WITHDRAW 70
 [THE BALANCE IS]
-10
```

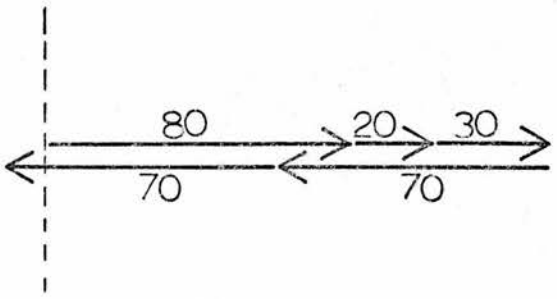


Figure 6.36. Money Movements.

The author also thought of this visual representation during the course of the session and, not realising that Jane had adopted this solution, started discussing it with Mary. The recording shows how throughout a long discussion with Mary, Jane was trying

unsuccessfully to break into the conversation to ask how she could have her procedure print out the value of the plotter pen's position.

At the end of the session, Jane explained her insight:-

"Well, I could not have done it with numbers. I find it much easier to use the plotter to do it and then go onto numbers."

This showed how Jane used the explicit visual model of integers to solve a new problem.

In the next session subtraction of integers was presented as the inverse of addition. That is to say, the problem of solving the following equation:-

$$(+230) - (-180) = X$$

was restated as

$$(-180) + X = (+230)$$

Jane then had to find the programming command which together with BACKWARD 180, gave a net effect of FORWARD 230. She found the correct answer, FORWARD 410, i.e. (+410). She was surprised that (+70) - (+160) could give the result (-90) because both the original numbers were positive.

When Jane had successfully subtracted a number of pairs of integers, she was asked if she could formulate a rule for these subtractions. She was able to do this and explained a rule which was derived from the drawing work. This consisted of putting the directed numbers 'tail to tail' for subtraction, in contrast to putting them 'head to tail' as in addition, see figure 6.37.

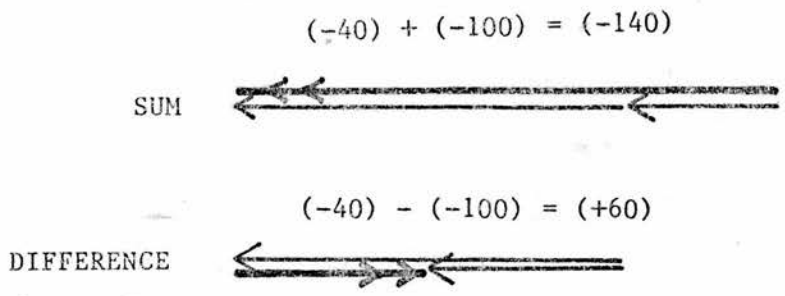


Figure 6.37. Combining Integers.

Although it was Jane who thought of the rule initially, Mary explained it to the author. She used the example (-40) - (-100).

Author: "Explain to me how it [the new rule] works."

Jane: "Well you start from the minus and you go backwards, well, it, you go backwards forty [draw (-40)]."

Author: "Ah, ok."

Jane: "And then you go back again to the, your centre point there."

Author: "Ok."

Jane: "And then you go back a hundred again [draw (-100)]."

Author: "Yes."

Jane: "And..."

Mary: "And then you find..."

Jane: "Then you come back to..."

Mary: "From the second point to the first point."

Jane: "That's right, uhu, that's what I am trying to say."

What Jane had done was to construct diagrams similar to those often used to represent the subtraction of vectors, see figure 6.38. But she had done this herself as a method of formalising the drawing work which was modelling subtraction. One must add that the suggestion that such a formalism be sought was that of the worksheet,

but the type of formalism was Jane's own.

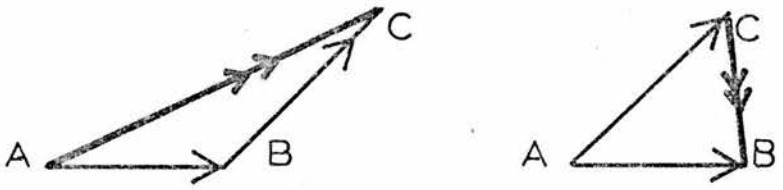


Figure 6.38. Combining Vectors.

Jane went on to explain about her earlier experience with integers:-

> "Yes, I can see now why, you know, you get pluses and minuses. But before when [the College lecturer] started talking about it, I didn't, you know...like he said 'a plus', like,'a minus and a minus equals a plus'. You see, I thought a minus and a minus must equal a minus, and he was going spare because I said how does, how was it."

Here she had misunderstood the lecturer's rule about multiplying integers by applying it to addition, possibly by misinterpreting the term 'and'.

Fibonacci Series

While the previous sections have concerned topics with which Jane had earlier been observed having difficulty, this subsection describes her work with a number series. This work arose as part of the programming teaching on recursion. It is included here because it is another example of how programming problems overwhelmed the mathematics.

The case in point concerns Jane's work with a recursive procedure which printed out a Fibonacci series. The definition of

such a series was embodied in the recursive procedure (below) but the definition could be more easily stated as:-

$$T(n) = T(n-1) + T(n-2)$$

where T(n) is the n'th term of the series.

The English statement of this is that 'each term of the series is the sum of the previous two terms'. In LOGO, the procedure which generated a Fibonacci series gave an explicit description of how terms in the series could be computed, which obscured the simple statement of the generating rule. This would not necessarily be the case for other programming languages. The text of the LOGO procedure was:-

```
DEFINE "FIBONACCI "NUMA "NUMB
10 PRINT VALUE "NUMA
20 FIBONACCI (VALUE "NUMB) (ADD VALUE "NUMA VALUE "NUMB)
END
```

Jane spent some time puzzling out how the control structure and the binding of argument values in the recursive calls produced the given series of numbers. But she was not able to give the author any concise statement about the 'rule' governing the series. Three months later, Jane returned to this problem but by now had forgotten how recursion worked so she could neither explain how the series was produced by the recursive procedure, nor could she give the rule governing the series. The next session Jane again spent some time on this problem and succeeded in explaining how the series was produced by the recursion saying:-

> "If I leave on a bad note [i.e. at the end of a session] it...you know...it preys on my mind all the time thinking about it."

This work had involved her in considerable effort because of the

complexities of the action of the recursion. But she still had little idea of the series rule. Three weeks later, the author asked her if she could see the pattern in the sequence 0,1,1,2,3,5,8... which she had studied before. Now she was able to give the next term, 13, and to explain how she arrived at it.

There appear to be two difficulties. Firstly the series was introduced as an example of the action of a recursive procedure. It had not been requested by Jane and had no supporting mathematical context. Secondly the syntax of LOGO obscured the rule rather than making it explicit.

6.5 SUMMARY

Jane's experience of learning mathematics through programming will be summarised using the framework set out in section 6.1.

Rigour and Explicitness

The evidence presented suggests that Jane had appreciated the value of the formal programming language, even if it was occasionally frustrating, because it enabled her to solve problems. Here the formality, though sometimes difficult to learn, was not seen as an expression of arbitrary rules but as a means of controlling a complex machine. Little work was undertaken to link the formality in programming explicitly to mathematical formality. But some pieces of conventional mathematical notation were explicitly linked to programming constructs e.g. parentheses and function notation.

Active Exploration

Various modelling facilities were provided for Jane to explore different mathematical systems e.g. Turtle Geometry, symmetry

transformations, and fraction and integer operations. Numerous examples were given of Jane grappling with geometric problems in the course of her programming work. Though the notion of formal proof and theorems were not explored, a number of informal proofs and theorems were explored, for example, the total turtle trip theorem.

Instances were presented of Jane asking, if not answering, mathematical questions which arose naturally out of her programming activity. For instance, there was the pattern of the sum of the interior angles of polygons and there was the search for a relation between the angles of an 'arrow'. Sometimes the analysis, though grounded in the programming activity, moved beyond it. An example here was Jane's schematic diagrams for subtraction of integers. But much of her work concerned the solution of programming rather than mathematical problems.

Key Concepts

Jane explored a wide range of important mathematical concepts including angles, functions, variables, integers. While the work with angle was extensive, not much was done with functions and variables. This was a shortcoming of the course of work rather than a difficulty associated with programming, or with LOGO. The work actually done on functions showed that programming could be very useful. On a number of occasions it was necessary for the author to make the explicit link between the programming work carried out and Jane's school-based mathematical knowledge. For example the relation between Turtle Geometry and use of protractors had to be discussed, as had the relation between the prefix function notation of her College worksheet and LOGO's prefix procedure notation.

This suggests that, if students are to gain insight into key mathematical concepts, then their programming course should explicitly formulate links between programming and the student's existing knowledge of mathematics.

Problem Solving

No attempt was made to observe changes in the students' ability to solve mathematics problems as a result of their programming work. Some evidence was gathered which supports the claim that programming provides a language for students to talk and think about their own problem solving and other cognitive skills. For example, after Jane had solved the problem of drawing a house, she reported that:-

> "Oh, huh, its an awful lot of paper just to do that. I suppose you learn by...by making the mistakes and having to put it all together again...I think you do. It really makes you think about it."

In the context of programming the idea of building on one's mistakes took on a precise meaning. It was not just a pious injunction for approved problem-solving behaviour. Two facets of programming are important here, its explicitness and the printed record of the programming process. The explicit nature of the task made Jane really "think about it". That is, a sloppy procedure would either not run or would produce the wrong effect. If she was to get anywhere she had to think out exactly what she wanted the computer to do. The other point concerns the printed record of the session. Even though she did not examine it to make a detailed analysis of how she had tried to solve the problem, she used it to make general remarks about the process, "an awful lot of paper". In principle,

examination of these dribble records by the subjects themselves could be used in much the same manner as video-tape is used in micro-teaching sessions (a review of research in this area is given by Brown, 1975). The students did use these records of the session to see how a problem had been solved earlier, and occasionally to think about how a session had progressed.

In the programming classroom, Jane was a most conscientious worker who tended to stick to suggestions for work given in the worksheets. She was also persistent and would work at a problem for a long time. Normally the author would leave her to get on with her own work unless she was obviously stuck, i.e. there was a long period of inactivity. Then he would go over to see what was wrong. He would also go over periodically to watch her and to talk to her about what she was doing. Jane tended to ask for help infrequently if the author was not sitting with her. However if he was sitting beside her, she would often let him take responsibility for what was done, asking him how to do things which she knew how to do herself. It was difficult for the author, to steer a middle path betwen being too helpful and being too distant. There were also the further considerations of gathering research 'data' in the form of her comments on her work. This problem was accentuated by having only a few students working at a time, typically one or two. Thus there was often no-one else the student could turn to except the author.

In questionnaire (2) Jane commented on the way she had been taught programming:-

> "I think its a good idea to leave us to think things out <u>for ourselves</u>, at least for us to make an attempt, and then, if we're wrong, to go over, and find out where we went wrong."

She was quite certain of the value of being left to solve problems herself. There was also an implied rebuke to the author who, especially in the early sessions, used to 'hover' near the students and be too quick to give help.

In questionnaire (3), Jane was asked what she considered 'the most useful thing in learning LOGO'. She replied:-

"Breaking down anything into smaller pieces especially in maths"

This point was reiterated in her answers to other questions in the questionnaire. For example:-

> "I think that sometimes I can look at a problem and split it down into bits -- before I tended to go head long without really looking at a problem --great difficulties following!"

Thinking About Learning

After the break during the summer vacation of 1976, Jane surprised herself because of the amount of programming knowledge she had retained. She explained why she was surprised:-

"...because first of all I have got a really hopeless memory for things. I find that...unless I keep going over a thing, I forget it very readily."

She went on to explain that it was things like programming which she would normally have expected to forget. When asked why she thought she had remembered it, she ventured:-

> "...possibly because you have got to think about it so much, and I suppose a lot of it does go in. Because you've got to...well like with this bit here [the work in hand] you have to, like when I was doing it before, you know, I had to keep going over it and over it again and you had to really puzzle out what when

you made mistakes. I find that helps a lot when remembering it...and I make a lot of mistakes and I have really got to puzzle out where I've gone wrong and then I'll remember it better. If I had done it perfectly the first time, it would have been in and out [of my head] in two seconds."

Jane explained at the end of the session that she had been nervous when she arrived, at the start of the session, because she expected to have forgotten everything and to have to go right back to the beginning. This fear was prompted partly by her belief in her own poor memory and also because of her experience after the previous large gap in her programming work, during teaching practice.

Two valuable lessons appear to have been learned. Jane clearly associated making mistakes, puzzling out those mistakes and learning and remembering. She also had a concrete instance of her ability to learn and retain this kind of subject matter.

She was also able to make some comparison between her experience of programming and that of a child learning mathematics. Thus from questionnaire (2) we have:-

"...its something new for us as learning maths is for a child. Many problems will be similar. <u>But I came to LOGO with a dread of maths.</u> That probably colours what I do--hopefully a child wouldn't feel like this."

And later on in a session:-

"Yes you do...I think you do...I think obviously you can appreciate a child's difficulty because when you come up against something here and it's really hard and you've got to work away at it...<u>and we find it hard, what must it be like for a child?</u>"

Here the value of the programming work was that it placed Jane

back in the position of being a 'learner' again in a very obvious way so that she could get some insight into the kind of difficulty which might be faced by the children whom she taught.

Attitude to Mathematics

A number of small increases in self-confidence were observed, related to Jane's increased understanding of particular mathematical topics rather than a wholesale shift in her belief in her ability to cope mathematically. The evidence for this has already been presented in the sections devoted to each topic. Some of it will be briefly repeated here, followed by evidence of her attitude to mathematics in general.

a) geometry

"I can sort of visualise an angle...but before it was dreadful."

"Its probably getting a bit easier now to think about angles. whereas before it was just a haze.

"..feel I could tackle this v. simply [in the classroom]- before I couldn't have attempted it without a lot of help."

b) Algebra

"I feel a bit more confident about these ones [composition of transformations]"

c) Fractions

"I know what it means now [equivalence of fractions], because I was never really clear exactly before."

"If I hadn't done this [work with fraction procedures] and some child had asked me...I wouldn't of...aha, I would say its just a rule"

d) Integers

"Yes, I can see now why you get pluses and minuses."

e) Attitude to mathematics in general

In answer to questionnaire (2), about whether she enjoyed using LOGO and why, Jane answered:-

"I enjoyed it at the beginning and still do. Why? (1) absorbing (2) I can 'see' how some of the ideas can be applied to school maths (very simple though) which is great as before maths was just a fog! (3) pulls a lot of stuff out of your mind that I didn't know was there! Obviously there's frustrating times, but, on the whole, I still enjoy it."

Her penultimate comment suggests that she has been pleasantly surprised by her ability to do programming. The "frustrating times" she alludes to were when she was trying to solve a problem and also when the direction and value of the whole enterprise was in question. Thus during one session she described that sometimes the programming work had been like "looking through a veil" and said that it had been a complete "fog". This seemed to be when the programming aspects of the work had overwhelmed the mathematical aspects.

In questionnaire (3), Jane was asked if her attitude to mathematics had changed. She replied:-

"...slightly more confident, but don't think anything could make me feel completely confident."

So it seems that the programming had helped a little, but she seemed resigned to the fact that mathematics was ever to be a cause of worry. She was also asked what she thought she had learned through LOGO. She replied:-

"(1) Tackling problems more confidently (2) Basic maths that I didn't know before (3) To be more aware of difficulties in a

topic eg. measuring angles."

It would appear that the work had helped her self-confidence a little. Her last statement indicated that she had become more aware of the difficulties of the children, but that she was beginning to be able to see causes for the difficulty which she could possibly tackle.

Disadvantages of Programming

Although the programming work did provide Jane with many opportunities for doing mathematics, there were also many instances where programming hindered her mathematics work. These hindrances may be divided into two classes. One was caused by the ready availability of the machine for experiments and the consequent focus on production of programming 'products'. The other was caused by the complexities of programming itself especially where an inappropriate set of primitives was used.

There were many instances where Jane perceived her activity as merely the construction of a particular procedure or the production of a given drawing. This is inferred from the following frequent characteristics of her programming behaviour:-

a) Lack of overall analysis of the problem and little preliminary planning.

b) Extensive use of successive refinement methods of procedure writing.

c) Failure to search for a more elegant solution once at least one solution had been found.

One example is her work on tessellations. The procedures she wrote were not especially complicated, so programming complexity was

not a problem. However she concerned herself essentially with producing pictures of tessellations rather than with an analysis of tessellation. These pictures were built up by drawing a little of the picture, debugging it, and then drawing a bit more. There was little prior planning. Once the picture had been produced, there was little attempt either to search for a more elegant solution or to think about the underlying mathematical constraints on the tessellation properties of different regular polygons.

Another example was the unsuccessful fraction project. Here Jane had a plan of attack, but her plan obscured the mathematical point which the procedure was intended to illustrate. This happened because Jane attended to the more obvious problem of how to draw the fraction pie-chart, rather than to the underlying problem of what such a pie-chart represented.

These difficulties must be attributed to the way Jane was taught to program and to the kind of question the mathematics worksheets asked. They illustrate a very real difficulty which faces those who wish to design mathematics courses based on programming. That is, the design of programming projects which successfully confront the student with mathematical issues without overwhelming her with irrelevant programming detail.

A second way in which the programming work hindered mathematical insight was caused by the complexities of the programming language. Here an example was the work done on Fibonacci number series. Most of Jane's effort went into finding out how the recursion worked and how the given series was produced by the procedure, rather than into understanding the series generating rule. This difficulty might have been reduced if Jane had studied a number of different

series-producing procedures. Then once recursion had been understood, she might have thought about the properties of the series themselves.

Many of what appear to be the most productive sessions involved Jane either writing very simple procedures or running procedures provided for her. These new primitives enabled her to study mathematical topics at an appropriate level of representation.

The programming sessions gave Jane the valuable opportunity to reveal and explore some of her mathematical difficulties. She succeeded in understanding some of the topics which had been puzzling her (for which she had been unable to get help within the College of Education). The practical benefit of the computer was that it enabled her to explore mathematical topics herself and experience the joy of success.

CHAPTER 7

IRENE AND MARY:

LEARNING MATHEMATICS THROUGH PROGRAMMING

The two case studies presented in this chapter describe incidents from the work of Irene and Mary. These case studies are less detailed than Jane's and only include incidents which contrast with Jane's experience.

Irene found programming difficult and unpleasant. She never mastered it sufficiently to get much benefit from it. She took a passive attitude towards her programming work and initiated few programming projects, but was content to work at what was suggested. She concentrated on the programming issues of her work and hardly ever explored the underlying mathematics.

Mary was quite different. She was much more curious about mathematics than either Jane or Irene and she used programming to explore mathematics to a much greater extent than the other two students. Not only did Mary notice mathematical patterns, as Jane had done, but she constructed explanations for those patterns. Her case study demonstrates that, once a student has understood a number of computational terms, these may be used as a vehicle for mathematical explanation without the necessity of running programs. It was sufficient to explain concepts in terms of hypothetical programs.

7.1 IRENE

Irene's case study concentrates on three aspects of her work. First of all she found programming and Turtle Geometry confusing and

frustrating. Secondly she adopted inefficient problem-solving methods (such as trial and error, and 'linear refinement') and concentrated on producing pictures rather than on understanding the geometric properties of the pictures. Thirdly she was given a poorly designed programming project which attempted to help her with a classroom difficulty concerning division.

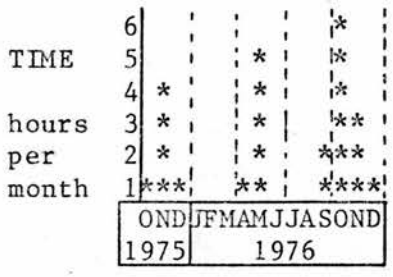


Figure 7.1. Irene's Programming Sessions.

Irene spent about 27 hours in the programming classroom. A table of her month by month programming activity is given as figure 7.1. The immediate contrast with Jane was in her relationship to the programming work. While Jane was conscientious, missed few programming sessions and telephoned if she did miss any, Irene had what appeared to be a much more casual attitude. She sometimes missed sessions but would come with excuses to a later session. She revealed later that one of the reasons why she missed sessions was because she found the programming so difficult. Missing sessions had the effect of making programming even more difficult for her to grasp. As a result Irene learnt less programming and covered far fewer mathematical topics than Jane. Unfortunately Irene came to many of the programming sessions on her own, whereas Jane and Mary often came together. This increased her difficulties because she did not benefit from the help of another student.

Frustration

Most of Irene's programming was carried out using the drawing devices. Irene, like Jane, had difficulty defining a procedure for a triangle though she did not make exactly the same kind of interior/exterior angle mistake as Jane. Irene tried to draw an equilateral triangle with three exterior angles of 45 degrees. This had the effect of drawing part of a regular octogon. She corrected this after a discussion with the author about 'walking around a triangle', the total angle turned and the angle turned at each vertex. Her difficulty over the way the turtle turned reappeared throughout her programming work.

After teaching practice, Irene had difficulty adjusting to the changes in the LOGO implementation. This meant that much of her effort was concentrated on formulating syntactically correct commands rather than on exploring mathematics. At this time she filled in questionnaire (2) which asked about her programming experience. Irene wrote :-

> "Yes I did and still do enjoy using LOGO. I find it fascinating yet confusing therefore I feel I have still got a long way to go before I get full meaning/understanding and benefit from it."

Her mixed feelings about programming, "fascinating yet confusing", were expressed with more force on further occasions. Her answer to the question about how the 'rotated house' picture could be drawn was much less explicit than Jane's, though Jane had the advantage of having just solved part of the problem prior to answering the questionnaire. Irene's answer was:-

> "Find a starting point, (turtle)-> connect and they [sic] begin giving LOGO various procedures."

Irene had a lot of difficulty when she came to draw a 'house' soon after. The session started with Irene listing various procedures she had defined in her first term including a procedure for drawing triangles of any size. When she started on the problem of drawing the 'house' these procedures were ignored. Again she defined a triangle procedure with angles of 45 degrees instead of 120 degrees as she had done before. By running the procedure she saw that it was incorrect. The author gave her some help in the method of editing procedures and then asked her about the angles of the triangle. She said that they had to be less than 90 degrees because it was an equilateral triangle. Irene asked if the triangle in the worksheet was meant to be equilateral. The author said that the illustration in the worksheet might have been a little inaccurate because he had drawn it with a ruler. Irene retorted:-

"I wish you would just get me a ruler, so that I could just draw with it too."

Her comment indicated a little of her frustration. She seemed to regard the programming as a means of producing a given picture 'product'. In this respect programming seemed highly constraining because it severely limited both what could be done and the means by which it could be achieved. From Irene's point of view the precise and rigorous formulation of procedures to draw the house seemed rather pointless, especially as the author had produced a similar picture with ruler and pencil for the worksheet.

The author continued his questioning about the angles of the triangle, asking whether they should be more or less than 45 degrees. This time Irene said she did not know, but added that she remembered being told about the angles of an equilateral triangle before. She

also added, a little accusingly:-

"You're not telling me anything, are you?"

The author explained that he believed it better that she experiment for herself because he had told her the answer before, and she had forgotten it. She then asked:-

"It's not something about three hundred and sixty or was it hundred and eighty?"

The author advised her to try a value different from 45 degrees even if it was just a guess. Eventually she suggested 120 degrees but could give no proper explanation. She said that an "obtuse" angle was greater than ninety degrees but that she had been trying to draw an "acute" angle. This was why she had tried values less than 90 degrees.

"It seems that, for to get it on that [the display turtle]...to get an acute angle, you've got to do more [give a number larger than ninety]."

The author then suggested she work with the button box and floor turtle because experiments could be conducted faster and because the rotations of the floor turtle were clearer than those of the display turtle. He asked her to draw a hexagon. She tried turns of 15, 5, 75 and finally 60 degrees in her attempts to produce a regular hexagon. She was beginning to get cross and so the author gave her more help by describing the total turtle trip theorem and the importance of 360 degrees. Irene spent the rest of the session trying to fit her triangle (now debugged) and square together to form a house using the display. She found it very frustrating, saying "I am getting exasperated."

The author suggested various tactics like clearing the screen

and re-positioning the turtle at the centre of the screen between each of her attempts. Eventually she abandoned her search for the correct set-up step between the square and the triangle, and decided to go home. The author discussed the session with her:-

Author: "Has it upset you?"

Irene: "No, I'm just annoyed."

The author tried to reassure her that she had learnt quite a lot and should come again soon to finish off the outstanding problem.

Irene: "...I just get myself into a...I feel so stupid...I know...I really...it makes me feel like, och, I just feel really thick."

Author: "Well that's, I mean, that's a shame. I don't want you to feel that way about it."

Irene: "Em, my mind goes exactly the same way as it does in maths, exactly the same."

Unfortunately the programming work had put her into exactly the same frustrating and debasing position she remembered from mathematics. She expanded on this point:-

Irene: "I put on the end of that thing [questionnaire (2)], I hope by the end of this course [the programming work] that my attitude towards mathematics will be one of pleasure and [she laughs] enjoyment."

Author: "But going by today's thing you are not terribly hopeful about it."

Irene: "Och yes, I enjoy it."

Irene's avowal that she "enjoys it" seemed to contradict her earlier comments, "I feel really thick", "so stupid" and "my mind goes exactly the same way as it does in maths". This contradiction

will be explored presently.

Drawing Pictures

Irene returned within a few days to finish solving the problem of drawing the 'house'. This time Jane was also present and was able to give her some help both with the language and with the drawing problem. Irene used the floor turtle and eventually, after trying various commands to orientate the turtle between the square and the triangle, she found the correct angle. Her search had been prolonged because the inaccuracy of the floor turtle masked the true solution. This led Irene to make two unnecessary attempts which differed from the true solution by 5 degrees. Her strategy had been predominantly one of visual inspection of the turtle's movement rather than an analysis of angles. When she had completed the drawing she did not know why the angle she had chosen was necessarily correct. The author helped her establish this by working through the procedure and by matching the turtle's movements to the individual commands.

Although she had drawn the triangle with the appropriate angle of 120 degrees, she still thought that its interior angles were 45 degrees. She seemed convinced that an equilateral triangle had angles of 45 degrees. The author explained that the angles inside an equilateral triangle had to be 60 degrees, given that she had used 120 degrees as the correct exterior angle. However the point was not pursued because she was pleased with her picture and was not much concerned to establish why it worked.

Irene's lack of analysis and her use of trial and error methods of solution diminished the mathematical benefit of the task. Irene saw her task as the production of a given drawing while the author

wanted her to examine the angle properties of the figures drawn.

Next session Irene explained that she was pleased to have solved the 'house' problem and that she had put the picture of the 'house' up on her wall at home. She had shown her father the picture and the teletype 'dribble' of the session.

> "My dad sort of looked at it [the picture] and then looked at the long piece of paper. 'My God,' he said 'you doing all that just for that wee thing.' You know they think it's all simple. But it's not really simple, not to work out how to get it. I don't think it's simple."

Like Jane, Irene had to reconcile the apparent simplicity of the 'product' with the complexity of the process of achieving it. The remarks of Irene's father and those of Jane are remarkably similar. Jane, however, saw some value in this process where Irene seems only to have seen the complexity.

About four months later, after the summer vacation Irene attempted further simple drawings. She wished to draw a diamond which she planned as a pair of triangles, see figure 7.2.

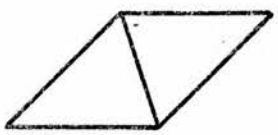


Figure 7.2. Diamond.

Her initial attempts were hampered by a system crash and so she was only able to produce a single non state-transparent triangle. She made many mistakes because of the long gap since she had last

programmed and confused procedure execution with procedure definition. She returned to the problem at the next session. This time her initial attempt produced part of an octogon. Then followed a long debugging session at which she adopted the inefficient 'linear refinement' strategy described in chapter 5, (see section 5.2). Once the 'diamond' was debugged, Irene started to incorporate it, as a 'leaf' and a 'petal', in the drawing of a 'flower' as suggested in a worksheet. The session was discussed:-

Irene: "It's a funny thing I think that you, you [herself] get a wee bit, a bit excited."

Author: "Yes"

Irene: "You know then it turns out somehow wrong and..."

Irene seemed surprised that she could get "excited" about such an activity. She explained that:-

"My brain isn't logical, that's why I don't get on with this." She continued:-

Irene: "...still just it makes you feel so stupid, you know, it's incredible."

Author: "Well I didn't, I didn't intend, I don't intend that."

Irene: "Well, I mean, I suppose...it's good anyway. It does make you...it really does make you really think. But I don't get..."

She had expressed these sentiments before about LOGO making her feel "stupid". The programming work was confirming her belief in her own inability rather than eradicating it. Her comment that "it's good anyway" was uttered without much conviction, in rather the same spirit that one might express oneself about some strenuous and unpleasant exercise precribed by a doctor.

Irene had mixed reactions to this session. She was both repelled and attracted by programming. She believed that programming was good for her but found it very hard and so felt "stupid". This feeling was, no doubt, accentuated by being observed by the author whom she knew could solve the programming problems. Her earlier surprise at getting "a wee bit, a bit excited" makes sense as her comment on her unexpected reaction to what was usually a rather unpleasant activity.

Irene continued to debug her 'flower' in the next session. She was adamant that the author should not stay near her, watching her. He adopted the policy of working elsewhere and returning at intervals to see how she was progressing.

Irene found the debugging difficult because she had no clear plan and was not sure which part of the 'flower' each of the procedures was intended to draw. Her habit of defining a new procedure, with a new name, as a method of debugging an existing procedure had produced a plethora of procedures with similar names and functions that was thoroughly confusing. The author helped her by getting her to explain what each procedure was to do and what the net effect of each procedure would be on the state of the turtle. Eventually with more help a 'flower' was produced and Irene asked:- "Will that do?" Irene seemed quite glad to have finished it. She seemed to look on this programming problem as an imposed task which it was her duty to complete to the satisfaction of the author. Irene was more concerned with the picture 'product' of the session than with either understanding why the program worked, or with constructing personally pleasing patterns. By way of comment on her performance, Irene noted that:-

"My brain needs a good shaking just now."

Failed Programming Project

Irene's classroom difficulty with with division was described in chapter 4. This concerned the distinction between sharing out objects among a given number of people and partitioning a set of objects into sub-sets of a given size. The author explained the distinction to her while they listened to the recording of the lesson in which the difficulty had arisen.

Irene was given a programming project which described the action of procedures to exemplify the distinction between sharing and partitioning. The project was unsuccessful because Irene found the procedures too difficult to write. Also the illustrations provided by the procedures (which were explained by the author) were not clear enough to characterise the distinction. Both procedures took argument values. The procedure illustrating 'sharing' worked as follows. Here the number of objects (12) and the number of recipients (4) was known:-

```
W: SHARE [T T T T T T T T T T T T] 4
[T T T]
[T T T]
[T T T]
[T T T]
```

This indicated that each of 4 people received 3 objects. The procedure to illustrate 'partitioning' worked as follows. here the number of objects (12) and the size of the sub-sets [T T T] was known.

```
W: BREAKUP [T T T T T T T T T T T T] [T T T]
4
```

Irene would probably have benefited more by manipulating real objects rather than by attempting to write procedures which manipulated symbols representing objects. The former would have been

closer to her classroom experience with the children and would have revealed the distinction more clearly.

The author was able to re-observe Irene teaching division. His explanations and the partially completed programming work produced only a small change in Irene's approach to teaching division. The children seemed to muddle sharing and partitioning more than ever.

Irene issued straws to the children. When the straws were shared out by the children, some counted the number of straws each had got and others counted the number of groups of straws produced by the sharing.

Irene: "Now I said, what did I say you were to do, David?"

David: "Share out so each person has three."

Irene: "Each person has three, no. We are going to share out sixteen straws to three people and I want these three people to have exactly the same amount of straws. So what, well what..."

pupil: "They will all get four."

Irene: "They will all get four, no. You've got five, you've got five and you've got five and there's one left over."

pupil: "Three remainder one"

The pupil counted Irene's three utterances of the phrase "You've got five" and looked at the three groups of straws on the table. This pupil's answer precisely characterised the partition/quotition muddle.

Irene: "Three remainder one, no. Why do you think it's three remainder one?"

pupil: "[inaudible]"

Irene: "Three bundles. You've got how many...in those bundles."

pupil: "It's five remainder one."

Irene: "Five, it's five remainder one."

At the end of the dialogue, Irene did direct the pupil's attention to the number of straws in each bundle as opposed to the number of bundles. However she did not make the point with any force. Irene's instructions still contained ambiguities, though she did not mix sharing terminology and partitioning terminology as strongly as she had done in the first lesson observed.

7.2 Mary

Mary gained much more mathematical benefit from her programming work than the other two students. She was mathematically more competent than they were. She scored higher in the mathematics test and displayed fewer difficulties in the classroom. Her case study shows how she used the drawing devices to create her own pictures. Like the other students, she wondered how learning to program would help her as a teacher. But she conducted a successful investigation of the mathematics of Turtle Geometry and made a number of personal mathematical discoveries. She used the computer to test hypotheses and as a source of interesting mathematical problems.

Mary studied functions by considering the code of a set of hypothetical procedures. She did not need to run these procedures, but she just hand-traced through them. This helped her to disambiguate the conventional function notation employed by the College of Education. She brought mathematical questions to the sessions which stimulated her. Her interest in them was fuelled by mathematical curiosity rather than by specific teaching needs. In many ways Mary had the least need of mathematical help and benefited

the most from it.

Mary spent about 50 hours programming. Like Jane she missed few sessions and covered a wide range of mathematical topics. A chart of her month by month programming time is given as figure 7.3.

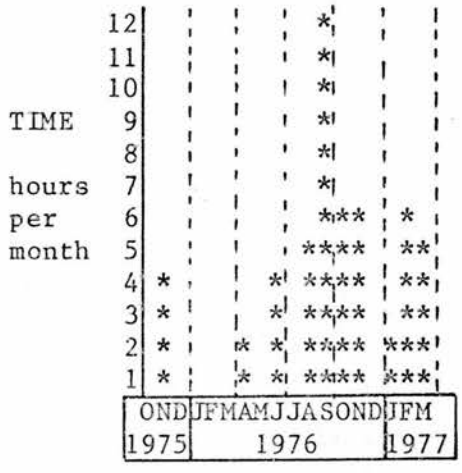


Figure 7.3. Mary's Programming Sessions.

From the start Mary worked quickly and self-confidently. Initially she was worried about asking questions but soon realised that the author welcomed them. Unlike Jane and Irene, she often noted down the author's replies and would often explain his answers back to him in her own words. Where Jane and Irene would listen passively, Mary would interrupt, ask further questions or summarise what had been said. Her whole approach to the programming work was more active. She was much more likely to mention bits of mathematics which either confused her or interested her. Of the three students, she was the most curious about mathematics, both for its own sake and because she needed to understand it in order to teach effectively.

Drawing Pictures

Mary had little apparent difficulty with the interior/exterior angle distinction and could produce correct turtle drawings more

easily than either Jane or Irene. After a long break, caused by teaching practice and by her College examination, she coped with the implementation changes, asked lots of questions and was easily able to define a procedure to draw an arrow.

Mary answered questionnaire (2) when she started programming again in June. Like Irene, her answer to the question about how the 'rotated house' should be drawn was not as explicit as Jane's:-

"You will need to get the turtle or other drawing devices. Give the commands. Ask Logo to define the commands."

When she came to the solve the problem of drawing a single 'house', she had difficulty with the distinction between procedure execution and procedure definition. She correctly used the angle 120 degrees as the angle to draw the equilateral triangle. She spent some time trying to fit her square and triangle together. Her difficulties were made worse because her procedures were not state-transparent and because she lifted the display turtle's pen between drawing the square and the triangle and forgot to put it down again before drawing the triangle. Eventually she produced a 'house' whose commands she then embodied in a procedure.

Mary then went on to use the 'house' procedure as a sub-procedure to draw a 'street' of 'houses'. In the debugging sequence for the 'street' she inadvertently produced a pattern of two 'houses' at right angles, as in figure 7.4a. She saw the possibilities of this and deliberately continued the pattern, see figure 7.4b.

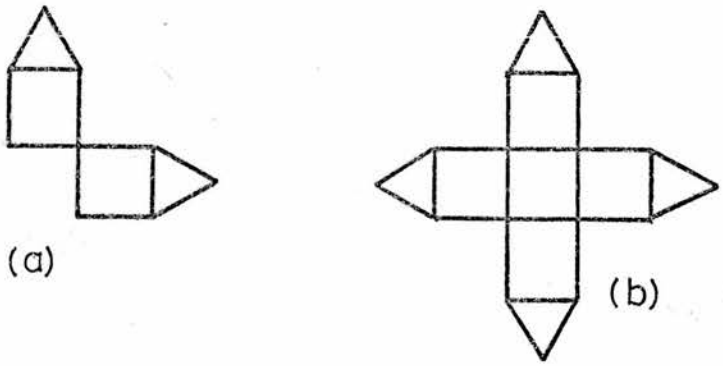


Figure 7.4. Pattern of Houses.

When the pattern was complete she defined a procedure, named SIGN, which drew a quarter of the pattern: a 'house' followed by a translation and a turn ready to draw the next 'house'. She then drew the whole pattern again on the screen by running SIGN four times. Then Mary returned to the problem of drawing a 'street of houses'. She realised that her procedure SIGN solved the problem of putting 'houses' together if she undid the effects of the turn command which rotated them. She successfully drew a row of four 'houses' (see figure 7.5) using the sequence:-

```
SIGN
LEFT 90
SIGN
LEFT 90
SIGN
LEFT 90
SIGN
```

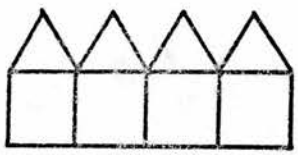


Figure 7.5. Street.

Mary then defined a procedure named STREET consisting of the pair of commands 'SIGN, LEFT 90' and spent the rest of the session drawing 'streets' of different numbers of 'houses' all over the screen.

Mary's behaviour in this session was in strong contrast to that of Irene who had also tried to draw a 'house'. Mary said that she liked to ask questions and did not mind whether the author was

watching her. She worked faster than Irene and made many mistakes. But she seemed much better at recovering from her mistakes, herself, whether they were syntactic or geometric. She was much more personally involved with the work having produced an interesting pattern as a by-product. She also went much further than Irene who had to struggle to produce even a single 'house'.

Next session Mary attempted to define a 'flower' following a suggestion in a worksheet. One of the procedures which she had to define was that for a rhombus (a 'diamond'). Her first attempt, like Jane's, had three angles of 45 degrees. However she immediately diagnosed the error and fixed it. But she did not succeed in completing the 'flower' because the super-procedures given in the worksheet expected her 'diamond' sub-procedure to be state-transparent which it was not. After spending some time attempting to draw the 'flower', she abandoned the problem and instead defined a long fixed instruction procedure which drew a 'range of mountains' similar to figure 7.6.

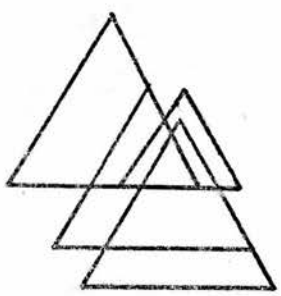


Figure 7.6. Range of Mountains.

Mary's reaction to programming and its setbacks was in marked contrast to Irene's frustration and feelings of incompetence:-

"When you go to LOGO you never realise how the time flies...and you forget about all the other worries."

Author: "Did you enjoy the session today?"

Mary: "Yes I think I did. It was, eh, no especially this achievement, that was good, you know, where I could find my own mistake and where I could correct it at the same time."

Mary had become quite adept at drawing with the turtle though she still completely misunderstood procedures which took arguments. In her next session she defined a hexagon procedure and went on to use this as a building block in a complex picture of a 'railway bridge', see figure 7.7. Again she was pleased with the session because she had solved a variety of problems herself, though the difficulty over arguments had not been resolved.

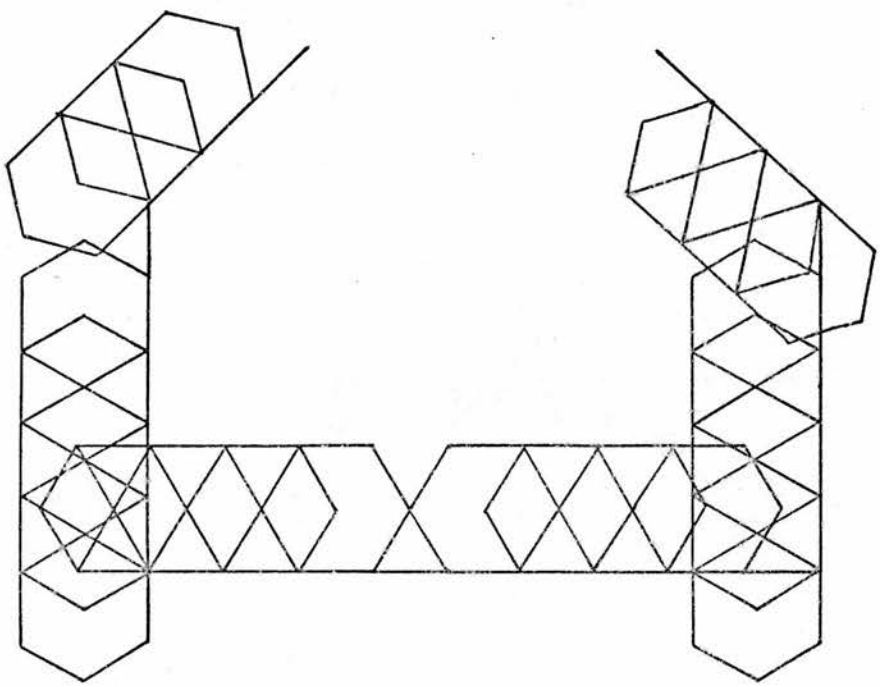


Figure 7.7. Railway Bridge.

Doing Turtle Geometry

The following sessions are described in some detail. They show how Mary's interest in the properties of polygons was stimulated by

her programming work. They also show how she explored and discovered relations among the angles of regular polygons. At the start Mary knew the sizes of the angles of familiar regular polygons such as squares. By the end she demonstrated an understanding of how squares shared a number of properties in common with other regular polygons. These sessions indicate that it is possible for students to explore Turtle Geometry to gain mathematical insights.

Mary drew a square and a pentagon whose angles were given in the worksheet and she was able to draw an octogon, having calculated its external angle herself. She tried to draw a 36-gon (exterior angle = 10 degrees) but it went off the display screen and so she could not see whether it closed or not. She correctly drew a 12 sided polygon (a dodecagon: exterior angle = 30 degrees) and then tried a second 12 sided polygon with an exterior angle of 20 degrees which was too large for the screen. This polygon would not have closed. She wondered why this last polygon was too large for the screen and then explained that a polygon with an angle of 20 degrees would be "broader" than one with an angle of 30 degrees and so would not fit on the screen.

Author: "Do you know how to decide whether the thing [polygon] is going to close up or not?"

Mary: "Er, no, this is what I was trying to discover, in fact."

The author suggested that she explore a bit further to establish the closure rule. But she asked how many degrees the turtle turned in a closed path. The author then showed Mary that a body rotated through 360 degrees when it moved round a closed path which did not cross itself. He asked her what angle it would turn through at each vertex of a square.

Mary: "It is ninety...ninety four times."

Author: "Right...ninety four times, yes ok, which comes to?"

Mary: "Three hundred and sixty."

Author: "Ok right. A pentagon?"

Mary: "That's five, er, three hundred and sixty divide by five."

Author: "Right, ok, that's it. That's the rule."

Mary then examined her 12 sided polygon with 30 degree angles, and her attempted 12 sided polygon (partially off the screen) with 20 degree angles and said.

"Aha, so twelve twos are twenty four...so two hundred and forty and so that wouldn't close up."

Mary's response to the worksheet instruction "draw a lot of different polygons" was to try to establish a rule for closure. Where Irene had taken a similar instruction quite literally and drawn some polygons, Mary concerned herself with the properties of the polygons. It seemed that now Mary understood the relation between the number of sides and the exterior angles of regular polygons. So the author suggested that, for homework, she think about how a circle, a five-pointed star and a six-pointed star might be drawn, see figure 7.8.

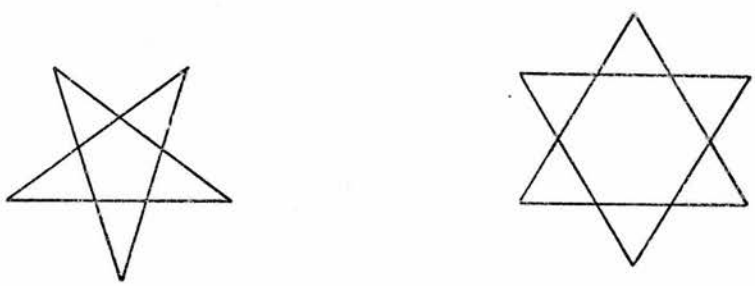


Figure 7.8. Two Stars.

Mary found this a stimulating problem and spent much time

wondering about it before the next session.

"I think I worked over it for three hours..."

When she arrived at the session, Mary immediately mentioned the "triangle within a triangle", the six-pointed star. She drew one by hand and labelled certain angles, incorrectly, as shown in figure 7.9.

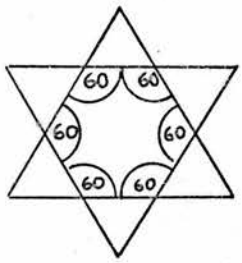


Figure 7.9. Incorrect Angles.

She reasoned that since there were six angles, each had to be 360/6 = 60 degrees. This was a misapplication of the total turtle trip theorem which she had explored in the previous session. Mary explained that her labelling led to a contradiction which she was about to describe when the author interrupted her to point out that she had labelled the angles incorrectly. He explained that the 360 degree total referred to the total exterior angle rather than to the total interior angle. Mary did not see the force of this until he asked her whether the angles she had labelled as 60 degrees looked more or less than 90 degrees.

> "Bigger than ninety degrees...aha...ah, now this is where...this is what I couldn't solve it, you see...and I didn't have a protractor...so I couldn't measure it. But I went and asked somebody at the office, I mean, where I am working at the moment [a vacation job] and he was thrown as well because...so if that

would be, that is ninety [she estimates the size of the angles], that would ninety...that would be forty-five degrees, eh, ah so that will be hundred and twenty degrees."

The author again pointed out that the total exterior angles of any polygon which does not cross itself is 360 degrees but that the total interior angles depended on the particular polygon. Mary told him the total interior angles for a square and a triangle correctly. The author started to draw up a table, see figure 7.10, with Mary's help. She now saw her mistake and was able to anticipate many of the entries.

"Because I asked, and they [the office staff] didn't know, you know, and so they said 'well you are a teacher, you ought to know'...and I was trying to puzzle that out...and I couldn't because I didn't have a protractor but it was stupid of me, I mean, I didn't look at the size of the angle."

No. OF SIDES	INTERIOR ANGLE	TOTAL OF INTERIOR ANGLES	EXTERIOR ANGLE	TOTAL OF EXTERIOR ANGLES
3	60	3*60 =180	120	3*120=360
4	90	4*90 =360	90	4* 90=360
5	108	5*108=540	72	5* 72=360
6	120	6*120=720	60	6* 60=360

Figure 7.10. Table of Polygon Angles.

Mary then filled in the table for the square and the regular pentagon. For the pentagon she calculated the exterior angle first and then the other angles, following the author's suggestion.

Mary then noticed that the interior angle total increased by 180 degrees for each extra side. Jane had noticed the same pattern but had been unable to explain it or make use of it. The author

suggested that Mary see if 'her law' continued to work for polygons with larger numbers of sides. She established that it did and happily announced that:-

"I have discovered something new, yes."

Mary was then able to give an explicit method for calculating all the angles of any regular polygon:-

Mary: "...a regular one, you would say that eh, what eh, number of [sides]...that is x, multiply by...no, sorry...x...three hundred sixty divide by x, ah, that would give you the exterior angle [exterior angle= 360/number of sides]"

Author: "Yes."

Mary: "And eh, that multiplied by the number x, I mean, no sorry...hundred and eighty take away the number you got would give you the interior angle [interior angle = 180 - 360/x] and if you wanted to find the total number of interior angle...that means one interior angle multiplied by the number of sides, that is x. [total interior angle = x(180-360/x)]"

Mary then returned to the contradiction which the author had interrupted earlier. She reasoned that if she labelled the angles as in figure 7.9 it would imply that the angles in the triangle in figure 7.11 would total at least 240 degrees but:-

"Funny, two hundred and forty but a triangle has only got hundred and eighty degrees."

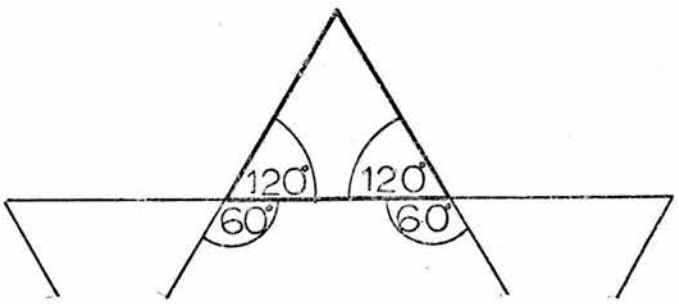


Figure 7.11. Incorrect Angle Values.

This was the contradiction that she had been unable to resolve. Mary then mentioned a difficulty in connection with circles. The basis of the difficulty was that she had not thought out clearly what she meant when she asserted that a circle had "got" 360 degrees. The difficulty was similar to her misunderstanding about the hexagon in the six-pointed star which she then thought had "got" 360 degrees. She now knew that a hexagon had a total of 720 degrees for it's interior angles. She said that if one then inscribed a hexagon inside a circle (see figure 7.12) it "changed" the number of degrees within the circle from 360 to 720.

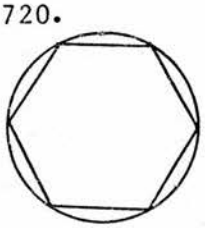


Figure 7.12. Hexagon Within a Circle.

The author suggested that she might explore this misunderstanding by trying to draw a circle using her polygon procedure. However, she was very keen to draw the five-pointed star and decided to tackle this first and leave the circle until later.

Mary did not apply her knowledge of polygons to the star problem. First she defined a procedure for the star which had unequal lengths of lines and unequal angles, which drew a shape as in

figure 7.13. She attempted to close the star by gradual modifications to the lengths and the angles in her procedure. She succeeded in completing an irregular closed star but was not satisfied with this as a solution. Her behaviour here contrasted with that of Irene and Jane, who were usually satisfied once they had produced the picture they were drawing.

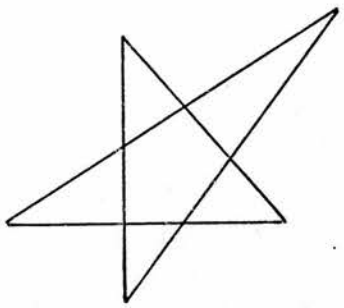


Figure 7.13. Failed Star.

The author asked her if she was trying to draw a regular star with all its lines the same length. She said that she was attempting this, although the lines in her procedure were of different lengths. A little later she had an insight into the problem:-

> "Oh, you know what, these angles in the middle should be the same throughout."

But she thought that the exterior angle total should be 360 degrees as it had been for the convex polygons considered earlier. The author showed her how 'walking around the star' took two complete revolutions, not one. She then was able to calculate the appropriate exterior angle (720/5 = 144) and drew the star using her general polygon procedure. She was pleased because now it was much clearer to her that the programming work was "getting into real mathematical things" whereas before she had wondered what the point of it was beyond teaching "logical" thinking.

Finally Mary considered the 'interior' angles of a circle. The author showed her that if she considered a circle as a polygon and followed the rules that she had already established then a circle had an infinite total for its interior angles. Mary tested this idea by drawing a 30 sided polygon which gave a reasonable approximation to a circle on the display screen.

Mary attended the next session with Irene. During a lull caused by a system crash, Mary told Irene about the polygon work from the previous session. Mary was very pleased to have solved the problem.

> "...there was a problem which I was supposed to solve and all of a sudden in the middle of the night it struck me why wasn't it working, you know. So I asked my sister, she just said 'Come on, throw that away in the middle of the night'. So I went to work in the morning and I asked the boys about it and one of them started playing a joke and said 'Why don't you ring to the Israel Embassy [about the six-pointed star] and they will tell you how to solve it...so I didn't get that solved, and then I came back here and I asked Ben [the author] about it. So eventually we got into this problem and we were able to solve it."

Then Mary showed Irene that the total of the interior angles of a polygon depended on the number of sides. Mary then discussed the polygon angle table with the author. She said that the total interior angle increased by 180 degrees for each extra side. She examined the sequence of individual exterior angles: 60, 90, 108, 120 etc. She asked:-

> "But I was wondering how much this [individual exterior angles] was going up by"

The author did not give her an explanation except to say it was an interesting problem. Mary continued to examine the table. Later she explained how she could use her rule, that the total interior angles increased in steps of 180 degrees, to calculate the angles of any regular polygon given the number of its sides.

Three weeks later, Mary suddenly mentioned the polygon table again. She re-explained that one could find the total interior angle of a decagon, say, by two methods. One method was to compute how many more sides a decagon had, compared to a triangle and then to add 180 degrees for each extra side to the interior total angle of the triangle.

TOTAL INTERIOR ANGLE = [(10 - 3) + 1] * 180

The other method was to divide the total exterior angle, 360 degrees, by the number of sides, 10, to find a single exterior angle. The interior angle was the supplement of this, and the total interior angle was a single interior angle multiplied by the number of sides.

TOTAL INTERIOR ANGLE = (180 - 360/10) * 10

The author asked Mary why her first method worked. At first she justified it in terms of the pattern she had discovered in the polygon angle table. The author persisted:

Author: "Yes, I know that, but why, but why is that when you add one more side you get exactly a hundred and eighty more degrees in the interior?"

Mary: "[after a pause] Oh is it because it makes, it makes an extra triangle."

The author suggested that she draw some polygons to make the point more explicit, which she did using paper and pencil, see figure 7.14.

Mary: "...so it is going up by each triangle."

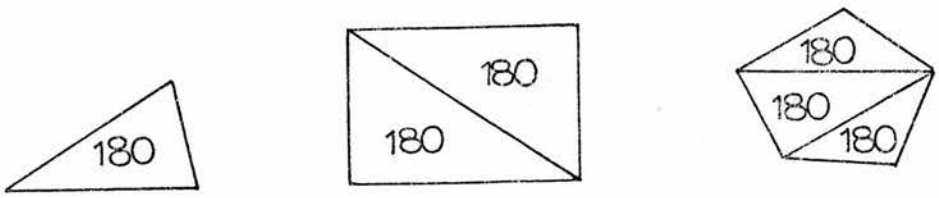


Figure 7.14. Adding Triangles.

The next session, Mary discovered a further property of regular polygons. She had defined procedures which repeatedly translated or rotated a basic shape such as a pentagon. She then considered the symmetry of various shapes such as the capital letters of the alphabet and the regular polygons. She explained that she understood reflective symmetry (because she could imagine a mirror placed over the figure) but found rotational symmetry harder to understand.

The author suggested that Mary consider the problem of an equilateral triangle in an identically shaped frame, see figure 7.15.

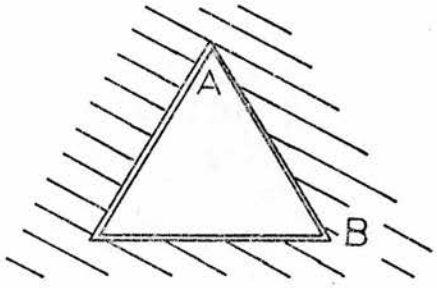


Figure 7.15. Triangle In a Frame.

Could the triangle be taken out, rotated in its own plane and then replaced back in the frame? Mary suggested that a 360 degree rotation would work and then saw that a smaller rotation would also do:-

Mary: "...if you took this point [A] and put it there [B] it will fit in."

Author: "Yes it will fit in, but how much do you have to turn the

whole triangle?"

Mary: "But how many degrees...is that one hundred and twenty?"

Author: "That's right."

Mary: "It's the external angle."

Author: "Yes."

Mary: "Ah...now how would you define that? How would you put it in simpler terms?"

Author: "For the children, you mean?"

Mary: "For children and for myself."

Mary was keen to express her insight in simple terms for her notes. She explored the relation between the exterior angle and the symmetry of a figure a little further. She found that a square agreed with her conjecture but that a rectangle did not, because one had to rotate a rectangle by 180 degrees to fit it back in its frame. She also found that that her conjecture did not work for an irregular pentagon. Eventually she understood that her rule worked only for regular polygons and expressed its essence unconventionally as follows:-

"The rotation symmetry is [the total] external angle divided by the number of sides of the regular polygon."

Next session, Jane was exploring the pattern in the polygon table but was unable to explain it. The author suggested that Mary show her. Mary tried to show Jane how polygons could be divided into triangles to illustrate the rule. Mary showed her the triangle and quadrilateral but forgot how to partition the pentagon correctly. Instead of the pentagon in figure 7.14, she drew one as in figure 7.16 which was divided into five triangles rather than three. The author had to intervene to help her with her with the drawing.

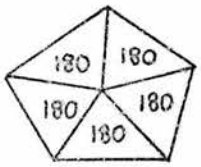


Figure 7.16. Wrong Division into Triangles.

Despite this final small setback, Mary's programming work with polygons had been most successful. From an initial contradiction about the angle properties of a star, Mary had moved on to a more general consideration of the angle properties of polygons. Then she had discovered a pattern among the angle properties of polygons and she had constructed an explanation for her pattern. Finally she had discovered a relation between the symmetry of a regular polygon and its exterior angle. The value of the programming work was that it stimulated the initial problem and provided an environment in which Mary could test her mathematical insights.

Where Irene had been overwhelmed by the complexities of programming, and where Jane had seen a pattern but had been unable to understand it, Mary had used the programming as a successful jumping off point for a piece of personal mathematics.

Coordinates

Coordinates were introduced to Mary as a method of positioning her drawings anywhere on the display screen. Her initial reaction was:-

"I have seen this in maps, but I haven't seen it in maths."

Like Jane, Mary was not familiar with the turtle's interpretation of negative arguments for FORWARD, BACKWARD, LEFT and RIGHT. So she expected to define different procedures to drive the turtle to different parts of the screen. She had little difficulty

in defining a procedure which would take the turtle from the centre of the screen to any point in the top-right hand quadrant. After some prompting from the author, Mary tried her procedure with its first argument value negative and then with its second argument value negative. She predicted that with two negative arguments, the turtle would be driven into the bottom left-hand quadrant of the screen. She then confirmed her prediction with evident satisfaction.

The value of this work was twofold. The procedure explicitly described how the ordered pair of coordinates was to be interpreted i.e. so much along and so much up. It also provided her with a means to position her pictures which she used many times thereafter with both positive and negative argument values. This provided a useful, personal application for negative integers.

Later in the term, Mary was observed teaching coordinates in school. The lesson had been inspired by a mathematics programme for schools on television. Mary and Jane listened to the recording of this lesson and Mary explained how her prior experience of coordinates in the programming sessions had helped her:-

Author: "Did you find it helpful to have done it [coordinates] here?"

Mary: "Yes it did because the teacher [the children's class teacher] was going to stop doing the workshop because she found it very, very difficult to follow"

(a little later in the conversation)

Author: "...So what I wanted to get at was, did you find that er, the work you'd done on coordinates?"

Mary: "Did help me, yes uhu."

Author: "So you had a strong idea about what was"

Mary: "Yes about what was happening there."

Author: "Uhu."

Mary: "Otherwise I think I would have been lost myself."

Mary continued by explaining that coming to LOGO was very helpful.

Mary: "Em, I mean this is what I said, I mean they asked me 'Why do you go there [to LOGO], is it voluntary?' and I said 'Yes it is' you know."

Author: "Yes, what, you mean here?"

Mary: "Yes coming here and I said 'I'm really glad that I've learnt quite a lot'"

Author: "Mm."

Mary: "In fact I sometimes find it is strain coming, you know, like when it is raining and so on..."

Mary explained how she had wondered initially what the purpose of the programming work had been, but that now she was much happier because she was tackling mathematical projects related to her school work:-

"No, I think especially this term, I mean after coming with my problems. Because I thought the first, you know, I thought the first two years when I had been coming [she has overestimated], I was wondering what is the point, you know. But I think this time it has been really worthwhile."

During this term and the next, Mary brought many 'problems' to the LOGO sessions. At this session she asked about the convention for the order of the coordinates in a pair:-

"Now just tell me Ben, I don't know, I have never figured out why you read the bottom number first [i.e. the x coordinate and then the y coordinate]."

The author replied that it was just a convention. At a later session, Mary recounted a related difficulty. Her pupils had constructed a scatter graph of their heights (a vertical measure) against their arm-lengths. Mary wanted to know whether the children's heights had to be represented on the vertical axis of the graph. She could not decide whether this was merely a matter of convenience or of greater mathematical importance.

Understanding Functions Computationally

This section shows how Mary learned about functions by considering a set of hypothetical procedures. She did not need to run these procedures, but just studied their properties. Mary took the same special mathematics course in the College as Jane. She also found that some of the topics, such as symmetry groups, were a little mystifying:-

> "...I can understand the simple reflection...but when he [the College lecturer] started the complicated stuff, you know, I couldn't understand. And when I asked him, you know...the way when they look at you as if that's easy...we are really stupid and ignore us...why did you do the course, you know."

Mary, like Jane, confused a transformation with the result it produced. She also had trouble filling up group tables and using the information in the tables to solve equations.

Mary had been given a worksheet on groups by the College. She found many of the qestions very difficult. Her exploration of one question in particular is now described because it shows how she and the author were able to make use of her prior programming experience. Several procedures were written in the course of this work but they

were not run on the computer. It was sufficient to rephrase the question in computational terms to make it more understandable.

The question from the College was to fill in a group table given:-

> The set of mappings {i,j,k,l} on real numbers defined thus:-
> i:x -> x
> j:x -> 1/x
> k:x -> -x
> l:x -> -1/x
> together with the 'multiplication' * where j*k means 'j, and then k'
> e.g. j*k:x -> 1/x -> -1/x

Mary did not fully distinguish between the mapping and the variable:-

"...so can you say the value of j is one third, one over x?"

Mary preferred to work with specific values of x, e.g. 3, rather than with the variable. The author showed her how the mappings could be represented as procedures. The name of the procedure was the name of the mapping and the variable x was the argument. The author defined the mappings i and j as follows:-

```
DEFINE "I "NUMBER
RESULT VALUE "NUMBER
END

DEFINE "J "NUMBER
RESULT DIVIDE 1 VALUE "NUMBER
END
```

Because the LOGO implementation only supported integers and not reals, the functions defined by the procedures had different domains and codomains to those defined in the College worksheet. This point was not mentioned to Mary. Since the procedures were not run on the computer the effect of the integer division 'DIVIDE' was not important (it would have rounded down the quotient to the nearest integer).

By the time the first two procedures had been written down, Mary was beginning to get the idea and anticipated the result of procedure 'K' as "minus the number", and that of 'L' as "result, subtract, result over, isn't it?".

```
DEFINE "K "NUMBER
RESULT SUBTRACT 0 VALUE "NUMBER
END

DEFINE "L "NUMBER
RESULT SUBTRACT 0 DIVIDE 1 VALUE "NUMBER
END
```

The author used the clumsy form 'SUBTRACT 0...' because there was no standard prefix form of MINUS in LOGO. The author then distinguished between the convention in the worksheet that 'j*k' meant 'j' before 'k', and LOGO's method of parsing commands. In LOGO the command:-

```
PRINT J K 4
```

meant apply 'J' to the result of applying 'K' to 4, in other words 'K' before 'J'.

In the conversation about the action of the mappings, Mary showed that she had not grasped the idea of composition correctly. She thought that 'j*k' applied to 3 meant 'do j to 3 and then do k to 3 as well'. Her difficulties were increased because she did not know the identities:-

$$1/(-x) = -(1/x)$$

$$1/(1/x) = x$$

Mary considered the composition 'i*1*j', but could not see how 'j' worked on the result of 'i*1'. So the author compared this composition to its equivalent LOGO form:-

```
PRINT J L I 3
```

where 3 was chosen as an arbitrary value for x. The author explained

how the LOGO command would be evaluated and how each procedure would get an input value and pass on a result, see figure 7.17.

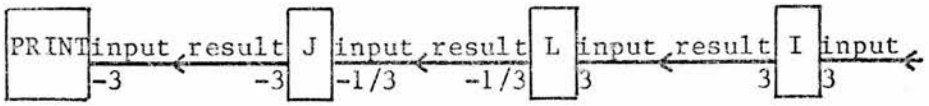


Figure 7.17. Result Passing.

Mary then tried to explain this back to the author. At first she did not distinguish properly between the mapping and its result:-

Mary: "...first 'I', I mean you are giving a number so 'I' is three."

Author: "Yes, 'I', well 'I' is a procedure and it's given an input."

Mary: "Right."

Author: "Ok."

Mary: "Well 'I', 'I' yeah, the input for 'I' is three."

Mary and the author continued to trace the execution of the command.

Author: "And J is told, whatever you're given, you put one over it."

Mary: "Ah, whatever you are given, put one over it, so that makes...right now I have got it."

Author: "Ok."

Mary: "I've got it. See, em, that is x, we got as far as that."

Author: "Yes, that..."

Mary: "So the work of...that was...the work of J is to put one over any number that was given to it."

Author: "Yes."

Mary: "Previously."

By the end of the dialogue Mary had formulated a clear personal,

and procedural, description of the mapping J which carefully distinguished it from the variable. Using the LOGO procedures, the author showed her how '1*j' was equivalent to 'k'. Mary again siezed the initiative and asked if she could explain this back to the author in her own words. She then considered the procedures which had been specified. Beside each one she wrote a brief note, as she described its action, see figure 7.18.

MAPPING	MARY'S NOTE
I	number
J	1
	number
K	-N
L	-1
	N

Figure 7.18. Mary's Notation.

Mary's personal notes on the action of the mappings were very similar to the formal descriptions given by the College. The procedural description had served an important function of explicating the original notation. Now that Mary understood the meaning of the mappings, the rather longwinded procedural representation could be abandoned for a more concise notation. The procedural representation had served a temporary purpose. This was not to replace conventional notation permanently but to explicate it.

Mary explained how pleased she was at the end of this part of the session:-

"Ah, I can see here that LOGO is really helpful making...I am pleased it has really worked for me."

Negative Integers and Subtraction

Mary explored the effect of negative integer arguments values on

the procedure for generating Fibonacci series described in Jane's case study. The important distinction between the meaning of the symbol for a negative integer and the symbol for the operation of subtraction was brought into sharp focus when she was puzzled by the behaviour of this procedure. In LOGO the operation of subtraction was called by running a prefix procedure, SUBTRACT. The symbol '-' was reserved for negative integers.

Mary ran the Fibonacci procedure (named SPAGHETTI) as follows:-

```
SPAGHETTI 200 -10
200
-10
190
180
370
550
920
1470
```

Each term was the sum of the previous two, where the two argument values, 200 and -10 were taken as the first two terms. But Mary was perplexed. It seemed to her that the terms were first decreasing in size and then increasing:-

"I see that, but the thing is I am trying to work out...if I give it, eh, a minus thing, why doesn't it go [on] deducting?"

Mary explained the 190 in the series as "two hundred take away ten" and the 180 as "a hundred and ninety take away ten". But she explained the 370 as "adding". That is, she interpreted the series as first subtracting and then adding rather than as consistently adding, initially with negative terms. She also noted that the relation between the terms could be expressed in a different way:-

"You can work it the other way round. You can say three-seventy take away a hundred and eighty [makes 190]." i.e. If

$I(n) = I(n-1) + I(n-2)$

then I(n) - I(n-1) = I(n-2)

Mary did not establish whether this applied to the early terms. She also explained that she really wanted the procedure to deduct 10, then deduct 20, then 30 and so on. The author suggested that she write such a procedure. He also suggested that she compare the action of the spiral generating procedure with the Fibonacci series procedure. The former would help to reveal the recursive structure and could be easily modified to produce the series which she wanted (by printing the lengths of the line segments). Mary's comments suggested that she might have been muddling up subtraction and negative integers. For example when describing the procedure she wished to define which deducted increasing multiples of ten, she said:-

"...I want, er, SPAGHETTI to give the numbers of negative, I mean the positive numbers deducting the negative numbers from the result"

Mary tried the procedure named SPAGHETTI again with two negative argument values:-

```
SPAGHETTI -200 -10
-200
-10
-210
-220
-430
-650
```

Mary explained its action:-

Mary: "In fact that is all right because, er, that is negative and negative, that's all right because negative and negative, it is er...you shouldn't add negative and negative numbers. So two hundred plus ten that is two hundred and ten. Two hundred and ten plus ten that is two hundred and twenty."

Mary had the idea of modifying the Fibonacci series procedure by replacing its addition by subtraction, i.e. so that it would generate the series:-

$$I(n) = I(n-1) - I(n-2)$$

She made the modification and tried out the new procedure, which she named FIBONACCI:-

```
FIBONACCI 200 -10
200
-10
210
-220
430
-650
```

The author asked her if she could explain why the third term was 210, but she could not. They traced the action of the procedure and then the author asked her more specifically:-

Author: "...so from two hundred subtract minus ten [200 - (-10)]."

Mary: "Aha."

Author: "Now do you know what happens if you subtract a negative integer?"

Mary: "From a positive."

Author: "Yes."

Mary: "I don't know with LOGO but it becomes one-ninety."

The author pointed out this mistake and explained the difference between:-

```
SUBTRACT 200 10
SUBTRACT 200 -10
```

He also pointed out that '-' conventially had two meanings and that the LOGO expressions above could be written:-

```
200 - 10
200 - (-10)
```

Mary then remembered a 'rule', "oh so that minus and minus changes into plus."

Mary decided to try out her new procedure FIBONACCI with pairs of positive numbers e.g.

```
FIBONACCI 2 20
2
20
-18
38
-56
94
-150
```

She spent some time trying to understand how the procedure produced the given results. Part of her effort was devoted to understanding argument binding in the recursion. But she also spent time trying to explain to the author why the terms of the series alternated in sign. This demanded that she subtract negative integers from positive. Thus she explained the expression 'SUBTRACT 18 -16' which had been evaluated during the run of her procedure as:-

"Take away, take away...so it becomes positive [34]"

having initially suggested '2' as the answer.

The value of this programming work was that it provided an interesting problem in which addition and subtraction of integers played a part. The programming language distinguished between the operation of subtraction and the symbol for a negative integer. It is unclear whether Mary's phrase, "take away, take away...so it becomes positive" is evidence that she had muddled this distinction or whether it was just a convenient rule of thumb for dealing with such subtractions.

When Jane's work with the Fibonacci series is compared to Mary's we find that Jane spent nearly all of her time trying to understand

how the recursion worked. In contrast, Mary investigated the recursion, the properties of the series and integer operations.

Mathematical Curiosity

Mary was more curious about arithmetic than either Jane or Irene. For example, she asked the author why the algorithms for addition, subtraction and multiplication all started on the right with the least significant digit, whereas the algorithm for division started on the left with the most significant digit. She had also been shown a quick method for long-division by a teacher. He showed her how she could change a division by 16 into two successive divisions by 4, so long as she carried out the appropriate computation on the two partial remainders. The example below divides by 24.

Division by 24 in one stage:

```
24)397
    16  Remainder 13
```

Division by 24 in two stages (24 = 6*4):

```
6)397
 4)66  Remainder 1
   16  Remainder 2
```

The particular example Mary cited was division by 16 (changing ounces to pounds). But she could not remember how to deal with the remainders. She realised that she could not just add them together since it gave an incorrect remainder in some cases. She asserted, incorrectly, that for division by square numbers such as 16 she could find the total remainder by adding the first partial remainder to the first factor but worried:-

"then I don't see any logic in it."

Jane, who was present, remembered doing similar calculations but

was not able to help. Mary explained that she and her sister had tried to find out how to do these calculations but had been unsuccessful.

At the next session, Mary was given a procedure to run so that she could try to solve this remainder problem for herself. The procedure, named BLOCKS, took two numerical arguments. The first value was divided by the second and the result of the division was illustrated by printing sets of 'X's and 'R's as follows:-

```
BLOCKS 13 3
XXX  XXX  XXX  XXX  R

BLOCKS 21 6
XXXXXX  XXXXXX  XXXXXX  RRR
```

Mary tried out this procedure on a number of divisions such as:-

```
BLOCKS 27 6
XXXXXX  XXXXXX  XXXXXX  XXXXXX  RRR
```

This illustrated that 27/6 was 4 remainder 3. Mary then compared this division by 6 with successive division, first by 3 and then by 2.

```
BLOCKS 27 3
XXX  XXX  XXX  XXX  XXX  XXX  XXX  XXX  XXX
```

then having counted the sets of 'XXX's

```
BLOCKS 9 2
XX  XX  XX  XX  R
```

The problem was to relate the single remainder 'R' in this last command to the remainder 'RRR' when dividing 27 by 6.

Once Mary thought she had solved the problem she wrote out the division on paper using a representation scheme similar to the 'X's and 'R's of the procedure. When she was satisfied that she understood the mechanism, she explained it to the author using her pencil and paper diagram.

Mary also explained the more complex comparison of '79/12=6 R 7' with '79/2/3/2' which she had tried earlier using the procedure BLOCKS.

Mary: "You are dividing it [79] by 2, you are getting thirty-nine blocks."

Author: "Yes."

(**)(**)(**).....(**)(**)(**) *

Mary: "Of each block containing two units."

Author: "Yes."

Mary: "And then getting one unit remainder."

Author: "Yes.

Mary: "Then you are dividing those thirty-nine blocks of two units in each block."

Author: "Yes."

Mary: "By three."

Author: "Yes."

[(**)(**)(**)] [(**)(**)(**)] [(**)(**)(**)]

[(**)(**)(**)].....[(**)(**)(**)] *

Mary: "So you are getting thirteen blocks."

Author: "Yes."

Mary: "Right."

Author: "Exactly."

Mary: "Ok and then those thirteen blocks have got six units."

Author: "Yes right."

Mary: "In them, you are dividing them by two."

Author: "Yes."

{[(**)(**)(**)][(**)(**)(**)]}.....{[(**)(**)(**)][(**)(**)(**)]}

[(**)(**)(**)] *

Mary: "So you are getting six whole blocks and then you are having a block left over which has got six units in it."

Author: "Yes."

Mary: "So six units and first one unit here, you're remainder is seven units."

The author could have explained the solution to Mary without use of a procedure. But by running the procedure and by attempting to relate its effects to the problem, Mary had the satisfaction of solving the problem for herself. The action of the procedure suggested a useful way of looking at the division. The procedure made one facet of the division process more explicit not through the body of its code, but throught its effects. Mary did not know how the procedure was defined. We may contrast this example with her work on mappings described in the last section. In both cases a procedure provided an explicit, but temporary descriptive system for a process. In the case of the mappings it was the text of the procedure which provided the explicitness. In this latter case it was the effects of the procedure. In each case Mary invented her own simpler notation to deal with the problem, once she had grasped its essentials.

7.3 SUMMARY

Mary's and Irene's experience of programming is summarised using the same framework which was applied to Jane's work.

Rigour and Explicitness

From the description of Mary's work it is clear that no attempt was made to teach her to derive theorems by rigorous argument from clearly defined axioms. The programming work undertaken did not

address the issue of 'rigour' in this sense. What it did do, however, was to illustrate the idea of 'explicitness' and the value of unambiguity in a formal language. Such 'proofs' as were constructed, e.g. about the interior angle properties of polygons, were not rigorous but highly plausible arguments.

While Mary was sometimes frustrated by LOGO's literal interpretation of her commands (as against 'understanding what she meant to say'), she was not overwhelmed by the details of programming in the way that Irene had been. This is not to say that Mary did not spend much of her time considering programming issues rather than mathematical issues. But she was able to cope with the programming and attack many problems at a more abstract, mathematical level.

Like Jane, Mary saw the value of forming explicit descriptions. In answer to the question "What do you think you have learned since doing LOGO?" from questionnaire(3), she replied:-

> "A [sic] break a problem in steps [sic]. To get thing [sic] clear in my mind and predicting the difficulties it might cause if my explaination [sic] is not clear."

A good example of the way that consideration of LOGO programming made a process more explicit was Mary's work on functions. The notation used by the College was concise and elegant.

```
k:x -> -x
```

But Mary's unfamiliarity with it meant that she did not fully appreciate the different status of the names 'k' and 'x'. Mary's greater familiarity with programming notation and with the action of procedures made the more verbose, procedural description a better vehicle for the idea. Once she had understood the idea she reverted to a notation very close to that of the College. The procedures,

which were never run, played a temporary but important role in making the idea of a mapping explicit.

On other occasions programming served to focus her attention on relevant issues. Thus work which started by drawing triangles and squares finished with an examination of the relation between the exterior angle of a regular polygon and its symmetry. The programming specification of regular polygons as sequences of 'FORWARD n, LEFT m' made such a progression possible by reducing the descriptions of the polygons to their bare essentials.

Irene found programming harder to learn than either Jane or Mary. She became much more frustrated by programming setbacks. The constraints imposed by the programming often seemed pointless to her, especially when she matched the simplicity of the picture 'products' of a session against the complexity of the process of programming the drawing. She planned her procedures less than either Jane or Mary and was less interested in understanding why a procedure worked.

Active Exploration

Mary's case study has shown how the programming work acted as a catalyst for mathematical exploration. Where Irene had struggled with programming, Mary was often able to attack a problem on a more abstract, mathematical level. Mary's search for explanations was more aggressive than Jane's, so she was more persistent in looking for the mathematics underlying some of the programming tasks.

Several examples were given where Mary was stimulated to ask mathematical questions arising from the programming work. For instance, her work on the angle properties of polygons was grounded in her early turtle drawing. Her analysis of the subtraction of

positive and negative integers was pursued in her attempts to understand the printed output of a series generating procedure. Her solution to the problem of the comparison between a single division and successive division by factors was prompted by her examination of the effects of running a procedure. Her work on coordinates and negative integers had the immediate benefit of enabling her to position pictures easily on the display screen.

Few instances were found where Irene asked mathematical questions about the programs or drawings she produced. Her behaviour contrasted very strongly with that of Mary, in this respect. Irene repeatedly grappled with the problem of understanding the way the turtle executed rotations but never seemed to sort it out fully. Most of her problem-solving was conducted more by trial and error methods than through an analysis of the problem. This had the effect that, when a solution was achieved, she had not benefited mathematically from the experience.

Key Concepts

Mary studied much the same range of mathematical concepts as Jane. Again the main emphasis was on geometry but useful work was done on the topic of functions and variables. This showed how even a 'pencil and paper' consideration of LOGO procedures and their execution could help clarify such issues as composition and the distinction bewteen functions and variables. In contrast, Jane had addressed these issues by matching a transformation to running a procedure, whose definition she did not know.

Irene's grasp of angle was still poor at the end of her studies. The partition/quotition distinction was only partially explored

through programming.

Problem Solving

Some of Mary's answers to questionnaire (3) suggest that she had understood the value of problem decomposition. In answer to the question, "What was the most useful thing in learning LOGO?", Mary replied:-

"Learning to work a problem step by step. When I could see the use of LOGO in connection with maths and how to solve problems using LOGO's procedure. Seeing the results of your procedures."

Another question was, "Since doing LOGO, have you found that the way you approach a problem is different? If so, in what way?" Mary responded:-

"Take a problem step by step and try and explain in different methods, if one method fails."

Mary's comments during the programming sessions show that she valued the opportunity to solve problems presented by the programming work. At the end of one session, Mary explained that what she had enjoyed was:-

"Now especially this achievement that was good, you know, em, where I could find my own mistake and where I could correct it."

Irene made only a few comments about her problem-solving strategy. On one occasion she explained that she was constructing a solution 'little by little'. Irene expressed general negative opinions about her own ability. These were not sufficiently specific to suggest remedial action on her part:

"My brain isn't logical, that's why I don't get on with this."

"My brain needs a good shaking just now."

When asked to make a comparison between her experience of programming and a child's experience of mathematics (questionnaire (2)), she concentrated on her feelings about the subject rather than on the methods of tackling it:-

> "...the first analogy is very much how I feel about LOGO. LOGO like maths is/was very alien to me."

Thinking About Learning

Like Jane, Mary used her experience of programming to think about problem-solving and learning. Mary explained that there were times when it was better to be told the answers but that:-

> "...to puzzle it out yourself and then getting the hint here and there...eh, helps you in fact...I think it leads you into...if you come across something else then you can link up the other past experience into it."

Later that term, Mary had an insight into the reciprocal nature of teaching when she was trying to work out the meaning of an error message from LOGO:-

Mary: "I can see it now."

Author: "You can see it now."

Mary: "Do you know what I can see? When the child doesn't understand the teacher and the teacher doesn't understand the child, the frustration starts."

The majority of Irene's comments about programming referred to her emotional reactions to it rather than to any insight into her own problem-solving behaviour. Although Irene said that programming made her "really think", the context suggested that on the whole this was an unpleasant experience rather than an enlightening one. She did

not look on her mistakes as constructively as either Jane or Mary and saw them as frustrating setbacks.

Attitude to Mathematics

Mary was more curious about mathematics than either Jane or Irene. As well as bringing her mathematical difficulties from the classroom or from College, she also brought bits of mathematics which had intrigued her. An example was the problem of comparing a long division with successive division by smaller factors. Her initial attitude to mathematics was different to that of Jane and Irene. They all, rightly, knew that they did not understand much mathematics. But Mary seemed to believe that she could learn to understand and she did not invest the subject with Jane's pessimism or with Irene's feelings of incompetence.

The programming work stimulated Mary's mathematical curiosity, as in the case of the six-pointed star, and more importantly, helped her to make personal discoveries. Thus after she had confirmed her conjecture about the pattern in the table for the angle properties of polygons, she said happily:-

"I have discovered something new, yes."

Mary consistently took a more active part in conversation with the author than either Jane or Irene. She would demand that he re-explain some point or she would try to explain the point back to him. She liked to put things 'in her own words' and to make notes. She did not have the same self-consciousness that Irene had and did not mind whether the author watched her or not, so long as he was available to give help when she wanted it.

At the beginning of the study Mary was rather deferential

towards the author and apologised for asking questions. But as time went on she felt more at ease and made several remarks about the style of the LOGO sessions. Thus at a session to which she brought no mathematical difficulties, she announced, "No confessions today". At another session she said that recounting her mathematical difficulties was like "telling sins in confession".

When working with Jane on integers, Mary commented, with some amusement, that the the author had been using "guided discovery". These comments reveal a rather less tense atmosphere than when the author was with Irene.

Irene gave no indications that her attitude to mathematics had changed. Irene made explicit comparisons between the unpleasant feelings she experienced while programming and those feelings she remembered from mathematics.

"...my mind goes exactly the same way as it does in maths, exactly the same."

Although Irene did eventually solve some of the programming problems, the experience seemed to remind her of earlier mathematical failures rather than to lead her to expect future mathematical successes.

Irene made few references to the way she was taught in the College of Education. Unlike Jane and Mary, Irene was not taking any extra mathematics course. Irene did not like being watched while she programmed and preferred either to seek help herself from the author or for the author to provide small amounts of specific advice. Unlike Jane she did not like the situation where she was attempting to solve a problem to which the author knew the answer, since it made her feel stupid. She also found that teaching practice lesson

observation made her nervous and very much more self-conscious about using the 'proper' mathematical vocabulary. She also felt "inhibited" by the one-to-one discussion with the author about the lesson recording. For her last lesson observation, the author gave Irene the tape first, so that she could listen to it and come prepared with her own comments. This seemed to help her a little though most of her comments were about her speech mannerisms than about the mathematics she was trying to teach.

Disadvantages of Programming

Much of Mary's work was concerned with programming rather than with mathematics. She made several references to the fact that most of the initial programming sessions had scant mathematical relevance but were concerned entirely to teach programming. In questionnaire (3), answering the question "What was the worst part of LOGO work?", she answered:-

> "a) When I could not see immediate value of LOGO work with the teaching of maths (2nd year) [she joined the study in her second Diploma year].
>
> b) When procedure [sic] made by me would not work out due to not enough inputs or language that LOGO did not understand."

Earlier she had made the same point during the polygon work when it became clear that something of mathematical benefit was happening:-

Mary: "It's starting to get more [inaudible word] now, it's getting into real mathematical things."

Tutor: "It is, isn't it? Do you think this is a reasonable way to explore mathematics?"

Mary: "Ehm, I think so yes, uhu. Because at first I couldn't see

the real meaning behind the...I mean what was I doing with LOGO all the time..."

Mary contrasted the early work, predominantly concerned with programming, with the later work concerned with her analysis of her own mathematical difficulties conducted through programming.

"No I think especially this term, I mean after coming with my problems, because I thought the first, you know, the first two years [she has overestimated] when I had been coming, I was wondering what is the point. But I think this time it has been really worthwhile."

Mary, like Jane, also attempted a number of topics, such as drawing fraction pie-charts and representing vectors as lines on the graph-plotter in which the mathematics was studied at the wrong level of representation. Mary's experience was similar to Jane's in this respect and it was not re-described in her case study.

Much of Irene's work was concerned with programming rather than with mathematics. The difficulties she had with the programming overshadowed much of her work. Very often she saw her task as the production of a particular picture, a 'house' or a 'flower' rather than a search for underlying geometric principles.

In more open-ended problems, such as exploring the action of a given procedure by supplying it with different argument values, Irene seemed content to supply a few values only and she formulated few questions about the action of the procedure. Her work with a spiral procedure did cause her some surprises, for example the production of a polygon when she supplied an increment of zero. But she only seemed to seek causes for these surprises when asked to by the author. She seemed to have little mathematical curiosity and never

experienced the obvious delight of moments of insight which Mary enjoyed.

CHAPTER 8

CONCLUSIONS

This thesis has investigated the problem of helping student teachers who had difficulties with mathematics. It has made three contributions to this field. Firstly, it has identified a number of different difficulties which student teachers had in explaining elementary topics in arithmetic. Secondly, it has given a realistic account of the advantages and disadvantages of enabling such students to study mathematics through computer programming, and has detailed a number of ideas for teaching topics from primary school mathematics. Thirdly, it has elaborated a strategy for teaching interactive programming which has been developed into a programming primer. it has provided data about the way novice, adult students learnt to program in LOGO.

8.1 MATHEMATICAL DIFFICULTIES

Previous investigations of the mathematical ability of student teachers have concentrated on their written answers to questions in mathematics tests. Such results give a depressing picture of students' abilities. This thesis has shown that the situation may be even worse. Students were observed while teaching and were interviewed about their lessons. They were found to have difficulty explaining basic arithmetic to their pupils. The students were generally able to perform the arithmetic computations which they had difficulty in teaching. So this kind of difficulty would not usually be detected by a written test. The main difficulties faced by the students were in illustrating, explaining and justifying mathematical processes and their applications. They knew the rules but did not

understand their meaning. Sometimes the students knew no justification for a particular rule, at other times they gave confusing explanations or mixed up two different applications of a rule. The students, consequently, suffered from a lack of mathematical self-confidence which sometimes extended to parts of mathematics in which the students were competent.

8.2 LEARNING MATHEMATICS THROUGH PROGRAMMING

The thesis has given a detailed account of how students learned, and failed to learn, mathematics through their programming activity. This has shown that programming as a method of learning mathematics has a number of benefits for such students. Programming encouraged a constructive view of mistakes and difficulties. The emphasis in the programming classroom was on understanding why the programs worked or did not work. There was little shame associated with a computer program which did not run correctly the first time. The need to make commands to the computer explicit forced the students to acknowledge and explore the topic which she did not understand.

A number of key concepts were given concrete illustration and explored by the students. Concepts with a dynamic element were especially well-suited to this method of exploration e.g. angle as rotation, functions as transformations and the transformation/state distinction. The students were able to explore a domain by observing how the primitives behaved and interacted. Some success was achieved in explaining mathematical concepts by reference to the code of hypothetical procedures and commands. Here the author and the students planned and hand-traced code but did not run it on the machine.

Students were able to pose and solve many problems in the domains defined by the primitives. Students gained a great deal of satisfaction by solving such problems. Mathematics was presented as an exploratory activity in which personal discoveries were possible. Turtle Geometry was used extensively. Students used this domain to ask mathematically interesting questions and to experience 'aha' moments of insight. Students found that they were capable of solving problems and usually enjoyed writing and debugging simple programs to draw pictures.

But there were difficulties in implementing a scheme based on programming. The teaching of the pre-requisite programming skills was over-emphasised so that it distorted the overall mathematical objectives of the work. At first the students did not see the relevance of their programming to their future work as teachers. The programming should have been been introduced in a mathematical context and the speicific links between programming and mathematical ideas pointed out from the start.

Initially some students were asked to represent mathematical processes in the notation of computer programs. But later in this study it was found that the major difficulty of the students was not their knowledge of how mathematical processes worked but their lack of understanding of why they worked and what they meant. For example, some students knew how to add, subtract, multiply and divide natural and rational numbers. But they did not understand the applications of these operations and they did not have explanations and illustrations for them which they could use in the classroom. Rewriting these processes as computer programs was a pointless activity when the work concentrated on the mechanics of the process

rather than on its meaning.

LOGO had a small range of data-types. This meant that it was necessary to write new primitives to study many mathematical topics. This was often an inappropriate task for the students to undertake because the majority of the resultant problems were concerned with programming rather than with mathematical issues. It proved more effective to provide students with these new primitives. This freed them to concentrate on the mathematical properties of the primitives.

Although the students learned how to write simple programs to control the turtle, they found complex programs involving symbol manipulation hard to plan and debug. This meant that many existing projects based on programming were too complex (e.g. Lukas et al., 1971). The students had limited time available in a crowded Diploma timetable. They did not see the value of investing large amounts of time in learning programming. Some students even found programming just as unpleasant and difficult as the mathematics it was supposed to elucidate.

There was a major difficulty in approaching mathematics through programming using drawing devices. The planning and debugging strategies observed among the students were their response to the ready availability of the computer and betrayed their preoccupation with the drawings as 'products' rather than with planning and analysis. Easy access to the computer encouraged them to plan and debug their programs at the console. Students were disinclined to analyse their programs once they appeared to work to see whether a more efficient or elegant solution could be formulated. Further attempts to teach mathematics through programming will have to guard against these tendencies and stress the importance of careful

analysis, away from the console, both before and after a program is written.

It was difficult to design programming projects which successfully confronted the students with their individual mathematical difficulties. Projects were unsuccessful when the mathematics was overwhelmed by the programming. For instance, asking students to write programs to draw representations of mathematical structures and processes was usually ineffective because the students concentrated on the visual properties of the representation rather than on the underlying mathematical properties. Again, programs which manipulated symbols representing objects were cumbersome and it seemed better (in hindsight) to ask the student to manipulate real objects. Representing integers as turtle movements was successful, however, because of the simple mapping between the two and because the integer operations could be investigated without writing complex programs. Ideas for teaching a number of other topics have been described, including fraction operations and symmetry transformations.

It was important to provide students with good teaching as well as good projects. Learning mathematics through programming did not mean that there was no need for help from a human teacher. The case studies gave numerous examples of where the author lost opportunities to exploit a situation mathematically. The case studies also showed that it was necessary for the links between the programming ideas and the students' existing mathematical knowledge to be made explicit. Communication between the student and the computer had to be supplemented by communication either with a teacher or with other students. Students should explain their programs in English to

someone as well as writing them in LOGO.

8.3 LEARNING PROGRAMMING

A programming teaching strategy based on a virtual machine and a primer to teach LOGO have been developed in conjunction with the parallel children's research project (du Boulay and O'Shea, 1976; du Boulay and O'Shea, 1978).

Evidence has been furnished on the coding errors of adult, novice, LOGO programmers. Cannara's (1976) analysis of errors made by children has been shown to be applicable to adults in broad detail. This confirms Statz's (1973) hypothesis that adults make the same kind of mistakes as children. It has been shown that many of the coding mistakes made by the students arose not from deep-seated misunderstandings but from failure to take all the syntactic rules into account. Other mistakes arose because the student changed context without informing the machine, e.g. by attempting to run a procedure that had just been edited before exiting from the editor. The main misconceptions of the students centred around variables and were concerned with the name/value distinction and with argument binding.

A number of planning and debugging strategies used by novices when tackling LOGO drawing problems have been described. Five strategies have been identified: 'direct-driving', 'linear refinement', 'incremental', 'exploratory' and 'partial top-down'. Such strategies reflected the students concern with getting an answer (e.g. drawing a picture) rather than with the process of problem solving or with the properties of the problem i.e. the students were product oriented. An example was given of the application of one of

these strategies, linear refinement, to a more complex problem involving transformation of symbolic data. This strategy produced a baroque and unwieldy partial solution. It suggests that emphasis on drawing problems may teach students inappropriate strategies for the solution of certain problems of data manipulation. This problem could be reduced by directing students through a more formal planning stage in their programming. However, early use of the drawing devices, as part of a teaching strategy based on a 'virtual machine' did allow the novice to write interesting programs easily at her first session. The drawing devices also enabled a variety of mathematical problems to be investigated visually.

8.4 CRITICISMS AND FUTURE WORK

Some teaching materials for particular mathematics topics have been developed. Many more are needed in order to implement a complete and coherent course in mathematics suitable for student teachers. The experiment described in the thesis was, necessarily, conducted with only a small group of students. This group was not a representative sample of student teachers and they were taught outside their College of Education in an University Research Laboratory. The author acted as the students' tutor, as an experimenter and as the evaluator of the work. One direction of future research would be to broaden the scope of the study by working with larger and more representative samples of students, by conducting the work within the normal course framework of a College or Department of Education, and by arranging that tuition and evaluation are undertaken by different people. Firmer conclusions could then be reached about the general applicability of learning

mathematics through computer programming.

The mathematical work undertaken by the students was largely determined by their mathematical difficulties. These difficulties were revealed in an ad hoc manner by classroom observation, by discussion and, to a limited extent, by the mathematics test which was administered. This could be improved in a number of ways. Methods for revealing difficulty could be more rigorous and the course work undertaken by the students could be better designed. A mathematics test plus a structured interview could be constructed to reveal students' difficulties. This would use the evidence in this thesis and the results of students' performance in mathematical tests, which have been reported in the literature. They would require the student, not only to answer written computations but would also demand that she explain topics to the interviewer. Kruteskii (1976), for example, provides a wide range of problems which might be adapted for this purpose. Examples of mathematical interview technique are given by Ginsburg (1976). The interview could be supplemented by micro-teaching sessions in which the student was expected to teach given topics to children. These methods would allow a mathematical profile of the strengths and weaknesses of each student to be built up.

In this experiment, the teaching of programming was initially over-emphasised and was taught with insufficient reference to mathematics. This could be improved in two ways. One way would be to redesign the programming primer so that it introduces programming in a mathematical context and teaches only those programming concepts which have an immediate part to play in a mathematics project. This would leave the emphasis on programming.

A different improvement, favoured by the author, would be to change the role of programming in the mathematics course. The emphasis should be shifted towards mathematics and away from programming. Programming should only be introduced in those parts of the course which merit it, and the programming projects should not distract attention from the mathematics. Effort is needed to develop sets of primitives which, like those for Turtle Geometry, produce a rich set of mathematically interesting behaviours without the need for complex programming knowledge on the part of the student. Such sets of primitives must have some relevance to the kind of mathematics which the students will eventually teach in primary school.

Further research should be conducted into the teaching of programming. This would establish whether the intuitive plausibility of basing teaching on a virtual machine is supported by controlled experiment. An improved programming primer could be written. This could use Turtle Geometry as an introduction but it would need to be more explicit about strategies for solving other classes of problem e.g. symbol manipulation problems. It should stress the importance of planning and analysis to counteract the tendency of the students to be satisfied with a drawing program which 'works' but which is an inelegant solution to the given problem. If the primer was to be used as part of a course in mathematics, the exercises, explanations and problems should be presented in a mathematical context.

Rather than widening the scope of the study, as described above, several factors would benefit from more detailed analysis. The description of students' mathematical difficulties showed how they had misunderstandings about elementary arithmetic. There exists a

large body of literature on children's understanding of arithmetic. Much of the work with children has investigated the relation between childrens' developmental level and the arithmetic concepts they understand. For adults the question of developmental level is not so prominent and their understanding of arithmetic could be profitably explored, for instance, with reference to the adult numeracy schemes currently being implemented (e.g. Riley and Riley, 1977).

APPENDIX 1

STUDENT DETAILS

GROUP 1

NAME	BEST MATHEMATICS QUALIFICATION
Anne	Mathematics '0' Grade
Betty	Arithmetic '0' Grade
Celia	Mathematics '0' Grade
Delia	" " "
Eve	Arithmetic '0' Grade
Fiona	Mathematics '0' Grade
Gail	" " "
Harry	" " "

GROUP 2

NAME	BEST MATHEMATICS QUALIFICATION
Irene	Arithmetics '0' Grade
Jane	" " "
Karen	" " "
Linda	" " "
Mary	(see note)
Nina	" " "
Olive	" " "

Mary had a Junior Leaving Certificate in Arithmetic from an Ugandan School. Her qualification has no exact Scottish equivalent, but it is probably equivalent to a Scottish Arithmetic '0' Grade.

APPENDIX 2

QUESTIONNAIRE (1)

PERSONAL MATHEMATICS (ARITHMETIC) HISTORY

I would like to know a little about your mathematical experiences at school and after. I have set out some questions to show you the kind of information I need. Please use the questions just as a guide. I would be very interested in any experiences, relevant to mathematics learning, that you particularly remember. Also if you can describe your present feelings about mathematics and the reasons why you so feel I would be very pleased.

PRIMARY SCHOOL

1. What kind of primary school did you attend? Did you change schools?
2. How many mathematics teachers did you have? Was much emphasis laid on mathematics?
3. Did you generally enjoy mathematics then? Why (or why not) ?
4. Can you remember the approach used? e.g. rote learning etc.
5. Can you remember any particularly enjoyable or nasty mathematical experiences?
6. Was there anyone in your family who was particularly good at mathematics?
7. If you did not enjoy mathematics by the time you left primary school can you remember if there was any strong turning point in your feelings for it?
8. What kind of things do you remember doing in mathematics in primary school?
9. Did you take an examination for secondary school? If so, what success?

SECONDARY SCHOOL

10. What kind of secondary school did you attend? Did you change schools?
11. Were you streamed for mathematics? If so which stream were you in? How did you feel about it?
12. Did you have a different mathematics teacher each year?
13. What sort of teachers were they and what was their general approach?
14. Did you enjoy mathematics in secondary school? Why (or why not) ?
15. Were there any parts of mathematics that you enjoyed? Were there any parts that you particularly disliked?
16. What mathematics exams did you take? And with what success?
17. Was there any particular thing about mathematics itself or the way it was taught that you did not like? (or like).

AFTER SECONDARY SCHOOL

18. Since leaving school and before joining Moray House, what sort of work have you done? e.g. raising a family, further

education, job etc.

19. Have you used any of the mathematics you learned at school apart from money transactions and the four rules?

MORAY HOUSE

20. Since joining Moray House has you attitude to mathematics changed significantly? If it has what caused the change?
21. Can you describe your present feelings about mathematics and about the thought of teaching it soon?

THANK YOU

APPENDIX 3

QUESTIONNAIRE (2)

LOGO QUESTIONNAIRE

1. Describe briefly how LOGO works. e.g.
   a) what sort of things it can do,
   b) how do you communicate with it,
   c) what are the major parts of the computer.
2. In trying to get LOGO to draw a complex shape such as  what are the main steps you would take towards a solution? (do not write any LOGO code)
3. Did you, and do you still, enjoy using LOGO? Why?
4. How would you convince another student teacher to come and learn LOGO?
5. If you were in charge of the LOGO course what improvements would you make in the way LOGO is taught?
6. Were you surprised at the amount of LOGO you had retained when you returned to it this term? What kind of things had you remembered? What kind of things had you forgotten?
7. Were you able to use any LOGO ideas or concepts or approaches while on teaching practice?
8. Even if the answer to question 7 was "no", can you see any direct use of any of the the LOGO in your future teaching?
9. Please comment on the following three analogies, pointing out where you feel that each analogy has some validity and where it breaks down.
   a) Yourself faced with the task of learning LOGO
      is like
      a child faced with the task of learning mathematics.
   b) Teaching the computer, that is writing procedures
      is like
      teaching children mathematics.
   c) My teaching you LOGO
      is like
      you teaching mathematics in school.
10. Please make any comments about LOGO, the way it is taught, what it is for etc, etc which you wish.

APPENDIX 4

QUESTIONNAIRE (3)

DEPARTMENT OF ARTIFICIAL INTELLIGENCE
UNIVERSITY OF EDINBURGH

LOGO QUESTIONNAIRE

These questions will be used to help me find out what mistakes I have made with the LOGO work. Please think about your answers and be honest. If you want to make any additional comments to any of the questions, please do.

NAME: DATE:

1. Do you think you have learned anything useful from the LOGO work?
2. Has the LOGO work helped you in any way with
   (a) your attitude to mathematics
   (b) your understanding of mathematics
   (c) your ability to teach mathematics.
3. What was the best part of the LOGO work?
4. What was the worst part of the LOGO work?
5. Did you enjoy the LOGO work?
6. Do you think your friends enjoyed the LOGO work?
7. Did you get any benefit from the recording of your lessons on teaching practice?
8. What practical difficulties did my visits to your lessons cause?
9. How did you learn LOGO. Put a tick by the things which helped you:
   (a) From your friends
   (b) From Ben
   (c) From the LOGO notes
   (d) From trying things out with the computer
   (e) Other ways - please specify
10. What was the most useful thing in learning LOGO?
11. How do you like working best?
   (a) On your own completely
   (b) With a LOGO teacher all the time
   (c) With a friend all the time
   (d) With a LOGO teacher available only if needed
12. Did your LOGO teacher help you
   (a) Too much
   (b) Not enough
13. Were your LOGO teacher's explanations
   (a) Too long
   (b) Too difficult
   (c) Too short
   (d) Clear
   (e) Confusing
   (f) Anything else - what?
14. If we could buy more equipment for the LOGO room what should we buy?

15. If we could move the LOGO room to Moray House, should we?
16. Put a tick by the things you did not like about the computer.
    (a) It broke a lot
    (b) It did not understand English
    (c) It got into a muddle over simple typing mistakes.
    (d) It gives unclear answers to questions
    (e) It takes too long to answer
    (f) Anything else - what?
17. What did you like about the computer?
18. Do you ever think of LOGO when you are trying to solve problems which are not connected with LOGO?
19. Do you ever find when talking to someone else (besides about LOGO) that you ever use some of the words you have learned in LOGO e.g. procedure, debugging?
20. Since doing LOGO, have you found that the way you approach a problem is different. If so, in what way?
21. Were you able to use anything learned in LOGO in any of your lessons?
22. What do you think you have learned since doing LOGO?
23. Has learning LOGO made any difference to you as a teacher?
24. Do you think LOGO is a good thing for people to learn?

[The following question was in two forms. 25a was used with those who were still attending LOGO sessions. 25b was used with those who had stopped attending LOGO sessions.]

25a. Would you like to continue doing LOGO?
25b. Please tick which reason(s) made you decide to discontinue LOGO work
    (a) Too much other course work
    (b) Lack of free time
    (c) Difficulty in getting to and from the University
    (d) Bored with LOGO
    (e) No clear direction in the work and benefits not obvious
    (f) Too hard
    (g) Not part of the accredited Diploma work
    (h) Other reasons - please specify?
26. I wish to continue this kind of work with other student teachers. What advice would you give me about
    (a) Improving the way LOGO is taught
    (b) Making the course relevant to the needs of the student teachers
    (c) Other changes in the course content
    (d) Making the visits to teaching practice lessons more useful
    (e) Ways of finding out from student teachers about the effects of LOGO
27. Do you have any other comments on the LOGO work?

[This questionnaire was developed from a questionnaire designed by Howe and O'Shea, 1976]

REFERENCES

ANZAI, Y. (1977) "Strategy Transformation Models In Problem-Solving". Carnegie-Mellon University.

ARCOUET, M. (1976) Personal communication.

ASSOCIATION OF TEACHERS IN COLLEGES AND DEPARTMENTS OF EDUCATION (1973) "Children Using Mathematics". Oxford University Press, London.

ASSOCIATION OF TEACHERS OF MATHEMATICS (1969) "Notes On Mathematics In Primary Schools". Cambridge University Press, London.

AUSTIN, H. (1976) "Teaching Teachers LOGO: The Lesley Experiments". A.I. Memo 336. Massachusetts Institute of Technology, Cambridge Massachusetts.

BARKER, P.J. (1975) "Lock Up The Black Box". Proc. IFIP 2nd World Conference 'Computers in Education'. Lecarme, O., Lewis, R. (Editors) North Holland, Amsterdam.

BARNES, D. (1973) "Language, The Learner And The School". Penguin Papers in Education, Middlesex.

BARNES, D. (1976) "From Communication To Curriculum". Penguin, Middlesex.

BARR, A., BEARD, M., ATKINSON, R.C. (1976) "The Computer As Tutorial Laboratory: The Stanford BIP Project". Int. J. Man-Machine Studies, 8, 567-596.

BJORK, L -E. (1975) "An Introductory Computer Programming Course And Some Of Its Effects On The Teaching Of Mathematics". Proc. IFIP 2nd World Conference 'Computers in Education'. Lecarme, O., Lewis, R. (Editors) North Holland, Amsterdam.

BRADY, J.M. (1974) "The Linguistics Of LOGO". Computer Science Memo CS4-5, University Of Essex.

BROWN, J.S., RUBINSTEIN, R., (1974) "Recursive Functional Programming For Students In The Humanities And Social Sciences". Technical Report No. 27, Dept. of Information and Computer Science, University of California, Irvine, California.

BROWN, G.A. (1975) "Microteaching: Research And Developments". In 'Frontiers of Classroom Research'. G. Chanan, S. Delamont (Editors), National Foundation For Educational Research.

BURSTALL, R.M., COLLINS, J.S., POPPLESTONE, R. (1971) "Programming in POP-2". University Press, Edinburgh.

CANNARA, A.B. (1976) "Experiments In Teaching Children Computer Programming". Technical Report No. 271, Institute for Mathematical Studies in the Social Sciences, Stanford

University, California.

CHARMONMAN, S., RALSTON, A. (1975) "Structured FORTRAN and the First Course In Computer Science". Proc. IFIP 2nd World Conference 'Computers in Education'. Lecarme, O., Lewis, R. (Editors) North Holland, Amsterdam.

CHASE, W.G., SIMON, H.A. (1973) "Perception in Chess". Cognitive Psychology, 4, 55-81.

DANIEL, J.S., VILLARDIER, L. (1976) "Deuxieme Rapport d'Evaluation sur Le Cours Permama Per 006 'Algorithmes et Programmation'". University of Quebec, Tele-Universite.

DAVIS, R.B. (1977) "Educational Applications Of Conceptualizations From Artificial Intelligence". Machine Intelligence, 8. E.W. Elcock and D. Michie (Editors). John Wiley.

DIENES, Z.P. (1973) "The Six Stages In The Process Of Learning Mathematics". National Foundation for Educational Research.

DU BOULAY, B., Emanuel, R. (1975) "LOGO Without Tears". D.A.I. Working Paper No. 11, Department of Artificial Intelligence, University of Edinburgh.

DU BOULAY, B., O'SHEA, T. (1976) "How To Work The LOGO Machine". D.A.I. Occasional Paper No. 4. Department of Artificial Intelligence, University of Edinburgh.

DU BOULAY, B. (1977) "Learning Teaching Mathematics". Mathematics Teaching, No.78, 52-57.

DU BOULAY, B., O'SHEA, T. (1978) "Seeing The Works: A Strategy For Teaching Interactive Programming". D.A.I. Working Paper No. 28. Department of Artificial Intelligence, University of Edinburgh.

DWYER, T.A. (1975) "Soloworks: Computer-based Laboratories For High School Mathematics" School Science and Mathematics, January, 93-99.

EDINBURGH REGIONAL COMPUTING CENTRE (1974) "Edinburgh IMP Language Manual". ERCC, University of Edinburgh.

EGGLESTON, J., GALTON, M.J., JONES, M. (1975) "A Conceptual Map Of Interaction Studies". In 'Frontiers of Classroom Research', G. Chanan, S. Delamont (Editors), National Foundation For Educational Research.

EISENBERG, T.A. (1976) "Computational Errors Made By Teachers Of Arithmetic: 1930, 1973". Elementary School Journal, 76, 229-237.

ELLIOTT, P.C. (1976) "Programming-An Integral Part Of An Elementary Mathematics Methods Course". Int. J. Math. Educ. Sci. Technol., 7, No.4, 447-454.

ELLIOTT, P.C. (1978) "Computer 'Glass Boxes' As Advance Organizers In Mathematics Instruction". Int. J. Math. Educ. Sci. Technol., 9, No.1, 79-87.

ERLWANGER, S.H. (1973) "Benny's Conception Of Rules And Answers In IPI Mathematics". Journal of Children's Mathematical Behaviour, 1, no.2, 7-26.

FEURZEIG, W., PAPERT, S., BLOOM, M., GRANT, R., SOLOMON, C. (1969) "Programming-Languages As A Conceptual Framework For Teaching Mathematics". Report no. 1889, Bolt Beranek and Newman, Cambridge, Massachusetts.

FEURZEIG, W., LUKAS, G. (1971) "Information Processing Models And Computer Aids For Human Performance". Report No. 2187. Bolt Beranek and Newman, Cambridge, Massachusetts.

FEURZEIG, W. (1977) "Notes On LOGO Language Specifications For Portable Implementation". Bolt Beranek and Newman, Cambridge, Massachusetts.

FISCHER, G. (1973) "Material and Ideas To Teach An Introductory Course Using LOGO". Technical Report No. 42, University of California, Irvine, California.

GARDNER, M. (1977) "The Concept Of Negative Numbers And The Difficulty Of Grasping It". Scientific American, 236, No.6, 131-134.

GINSBURG, H. (1976) "Learning Difficulties In Children's Arithmetic: A Clinical Cognitive Approach". Proc. Research workshop 'Models for Learning Mathematics', Osborne, A.R., Bradbard, D.A. (Editors), Georgia Center for the Study of Learning and Teaching mathematics, ERIC/SMEAC.

GOLDSTEIN, I., PAPERT, S. (1976) "Artificial Intelligence, Languages And The Study Of Knowledge". A.I. Memo 337. Massachusetts Institute of Technology, Cambridge, Massachusetts.

GREEN, T.R.G., SIME, M.E., FITTER, M. (1975) "Behavioural Experiments On Programming Languages: Some Methodological Considerations". MRC Memo No. 66, Social and Applied Psychology Unit, Sheffield University.

GREEN, T.R.G. (1977) "Conditional Program Statements And Their Comprehensibility To Professional Programmers". J. Occupational Psych., 50, 93-109.

GRIFFITHS, H.B., HOWSON, A.G. (1974) "Mathematics: Society And Curricula". Cambridge University Press, London.

GUILFORD, J.P. (1957) "Fundamental Statistics In Psychology And Education". McGraw-Hill, New York.

HAMPTON, J.A. (1976) "Service Mathematics Teaching Using A Computer-Based Laboratory". Paper to 'Computers in Higher

Education' conference, Loughborough.

HAYLOCK, D.W. (1977) "Achievement In Mathematics Of Students Preparing To Teach In Junior Schools". Mathematics In Schools, 6, no.4, 22-23.

HILLIS, D. (1975) "Slot Machine: Hardware Manual". LOGO Working Paper No. 39, Massachusetts Institute of Technology, Cambridge, Massachusetts.

HOC, J.M. (1977) "Role Of Mental Representation In Learning A Programming Language". Int. J. Man-Machine Studies, 9, 87-105.

HOWE, J.A.M., O'SHEA, M.M. (1976) "Computational Metaphors For Children". D.A.I. Research Report No. 24, Department of Artificial Intelligence, University of Edinburgh. To appear in Naturliche und Kunstliche Intelligenz, F. Klix (Editor), Deutsher Verlag, Berlin.

JOINT MATHEMATICAL COUNCIL OF THE UNITED KINGDOM (1977) "Desirable Qualifications For Teachers Of Mathematics In Primary, Secondary And Further Education". Shell Centre for Mathematical Education, University of Nottingham.

KAHN, K.M. (1975) "A LOGO Language System". LOGO Working Paper No. 46, Massachusetts Institute of Technology, Cambridge, Massachusetts.

KEMPLAY, J., DU BOULAY, B. (1976) "Turtle Memo No. 3" D.A.I. Technical Memo. No. 3, Department of Artificial Intelligence, University of Edinburgh.

KENNEDY, T.C.S. (1975) "Some Behavioural Factors Affecting The Training Of Naive Users Of An Interactive Computer System". Int. J. Man-Machine Studies, 7, 817-834.

KERSLAKE, D. (1974) "Attitudes, 1974". Mathematics Teaching, 68, 47-48.

KIEREN, T.E. (1975) "On The Mathematical, Cognitive And Instructional Foundations Of Rational Numbers". Paper from Research Workshop "Number and Measurement", R.A. Lesh, D.A. Bradbard (Editors), Georgia Center for the study of Learning and Teaching Mathematics, ERIC/SMEAC.

KRUTESKII, V.A. (1976) "The Psychology Of Mathematical Abilities In Children". Translated by J. Teller. University of Chicago Press, Chicago.

LEDDY, T. (1977) "Mis-Directed Numbers". Mathematics Teaching, No.78, 26-29.

LEESON, C.M., JAWORSKI, J., (1975) "The Hertfordshire Computer-Managed Mathematics Project". Proc. IFIP 2nd World Conference 'Computers in Education'. Lecarme, O., Lewis, R.

(Editors) North Holland, Amsterdam.

LUKAS, G., et al. (1971) "LOGO Teaching Sequences". Volume 3, Report No. 2165, Bolt Beranek and Newman, Cambridge, Massachusetts.

LUMB, D. (1974) "Student Teachers And Mathematics". Mathematics Teaching, 68, 47-50.

LUMB, D., CHILD, D. (1976) "Changing Attitudes To The Subject And The Teaching Of Mathematics Amongst Student Teachers". Educational Studies, 2, no.1, 1-10.

MATTHEWS, G., BAJPAI, A.C. (1977) "Report On Colloquium On Mathematics Education". Int. J. Math. Educ. Sci. Technol., 8, no.3, 365-367.

MAYER, R.E. (1976) "Some Conditions Of Meaningful Learning For Computer Programming: Advance Organizers And Subject Control Of Frame Order". J. Educ. Psych., 68, No.2, 143-150.

McARTHUR C.D., (1973) "LOGO User's Guide and Reference Manual". Bionics Research Report No. 14, Department of Artificial Intelligence, University of Edinburgh.

McARTHUR C.D. (1974) "EMAS LOGO User's Guide and Reference Manual". Occasional Paper No. 1, Department of Artificial Intelligence, University of Edinburgh.

McCARTHY, J. (1969) "LISP 1.5 Programmer's Manual". M.I.T. Press, Cambridge, Massachusetts.

MILLER, L.A. (1974) "Programming By Non-Programmers". Int. J. Man-Machine Studies, 6, 237-260.

MILLER, M.L., GOLDSTEIN, I.P. (1976) "Parsing Protocols Using Problem Solving Grammars". A.I. Memo No. 385, Massachusetts Institute of Technology, Cambridge, Massachusetts.

MILNER, S. (1973) "The Effects Of Computer Programming On Performance In Mathematics". Paper presented at American Educational Research Association, ERIC Reports ED 076 391.

MINSKY, M. (1970) "Form And Content In Computer Science". J. Assoc. Comput. Mach., 17, April.

PAPERT, S. (1971) "Teaching Children Thinking" A.I. Memo 247. Massachusetts Institute of Technology, Cambridge, Massachusetts.

PAPERT, S. (1972) "Teaching Children To Be Mathematicians Vs. Teaching Children About Mathematics". Int. J. Math. Educ. Sci. Technol., 3, 249-262.

PAPERT, S. (1973) "Uses Of Technology To Enhance Education". Proposal to the National Science Foundation, Artificial

Intelliegence Laboratory, Massachusetts Institute of Technology, Cambridge, Massachusetts.

PERLMAN, R. (1974) "TORTIS: Toddler's Own Recursive Turtle Interpreter System". LOGO Memo No. 9, Massachusetts Institute of Technology, Cambridge, Massachusetts.

POLYA, G. (1957) "How To Solve It". Doubleday, New York.

RAY, S.P. (1975) "Some Factors Affecting Recruitment To Main Mathematics Courses In A College Of Education". Unpublished M.Ed. Thesis, University of Newcastle.

REES, R. (1973) "Mathematics In Further Education: Difficulties Experienced By Craft And Technician Students". Hutchinson Educational.

REES, R. (1974) "An Investigation of Some Common Mathematical Difficulties Experienced By Students". Mathematics In School, 3, no.1, 25-27.

RILEY, T., RILEY, C. (1977) "Teaching Numeracy: A Manual For Tutors". National Extension College, Cambridge.

ROMAN, R.A., HELLER, J.I. (1975) "LOGO: A Student Manual". Learning Research and Development Center, University of Pittsburgh.

SCHMITT, A. (1975) "Simulative Transfer Of Mathematical Concepts By Interactive Programming". Proc. IFIP 2nd World Conference 'Computers in Education'. Lecarme, O., Lewis, R. (Editors) North Holland, Amsterdam.

SCHOOL MATHEMATICS PROJECT (1974) "Computing In Mathematics - Teacher's Companion From 11 To 16". Cambridge University Press, London.

SHNEIDERMAN, B. (1977) "Measuring Computer Program Quality And Comprehension". Int. J. Man-Machine Studies, 9, 465-478.

SKEMP, R.R. (1975) "The Psychology Of Learning Mathematics". Penguin, Middlesex.

STATZ, J. (1973) "Syracuse University LOGO Project: Final Report". Syracuse University, New York.

STONES, E., MORRIS, S. (1973) "Teaching Practice: Problems And Perspectives". Methuen, London.

TAIT, K., HARTLEY, J.R., ANDERSON, R.C. (1973) "Feedback Procedures in Computer-Assisted Arithmetic". Br. J. Educ. Psych. 43, 161-171.

WEYER, S.A., CANNARA, A.B. (1975) "Children Learning Computer Programming: Experiments With Languages, Curricula And Programming Devices". Technical Report No. 250, Institute for Mathematical Studies in the Social Sciences, Stanford

University, California.

WILLIAMS, E., SHUARD, H. (1970) "Primary Mathematics Today". Longman, London.

WILLIAMS, J.D. (1971) "Teaching Technique In Primary Mathematics". National Foundation for Educational Research.

YOUNGS, E.A. (1974) "Human Errors In Programming". Int. J. Man-Machine Studies, 6, 361-376.

The work reported in this thesis is my own.

The thesis has been composed by myself.

J.B.H. du Boeuf